SIGS

Java 2: Ready for Prime Time p. 6 • AOL/Netscape Deal to Boost Java, Jini p. 12

February 1999
Volume 4, Number 2
$4.95 US/$7 Can/$4.95 Int'l
49390

Java™ REPORT

THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT • www.javareport.com

Internationalization

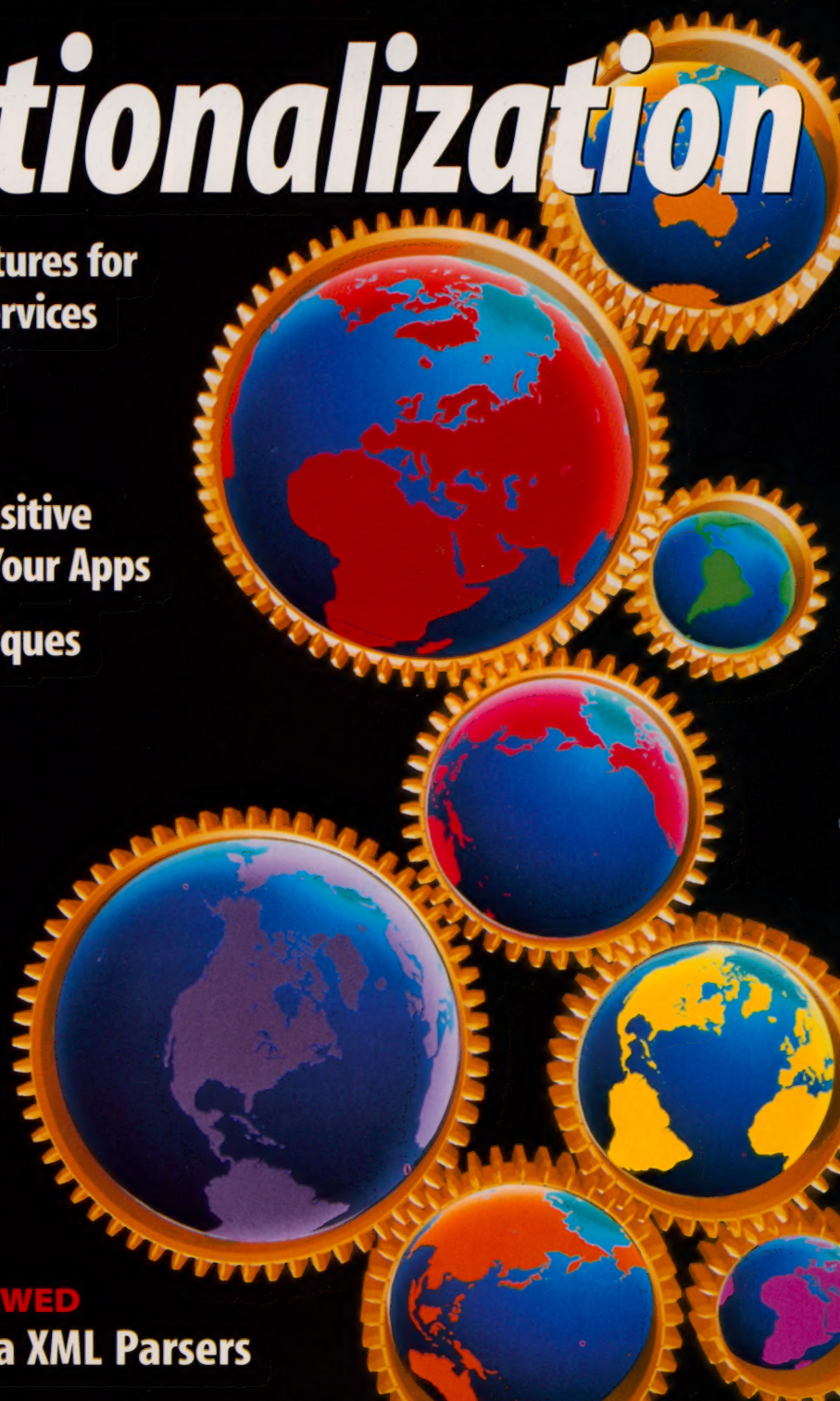
- Java 2 Platform's New Features for Mid-Level International Services
- Tips for Developing Multi-Language Applications
- How to Add Language-Sensitive Sorting and Searching to Your Apps
- Are We Ignoring OO Techniques in Favor of Tools?

Supercharged Strings: Using Special Properties to Improve Code Efficiency

Building Durable Java APIs

Java Printing Demystified

Build Your Own Search Strings Using JavaScript



0 71896 49390 0 02
49390

REVIEWED
7 Java XML Parsers

© 1998 4thpass. All rights reserved.

4thpass

SourceGuard3.0

Smart Bytecode Protection

[www.4thpass.com]

Microsoft

Where do you want to go today?®

With Visual J++ you can:

Visually design Windows and Web applications

Quickly create server-side business components

Easily debug and deploy multi-tier solutions

NEW VISUAL J++ 6.0

Finally, a Java language tool for the **PLATFORM** you see most.

Microsoft's Visual J++® 6.0 Professional Edition development system is the only tool for Java optimized for Windows® and the Web. This means you can now create Java applications offering the functionality, performance, and integration with existing software your users demand. Take advantage of Windows Foundation Classes (WFC), an object-oriented framework that simplifies and unifies Windows-based and DHTML programming. Go on to create COM components that integrate with existing business systems. Then instantly deploy applications to any remote file or Web server, making distribution and configuration no big deal. And it's all in a highly-productive, visual RAD environment. To preview this new tool, especially designed for the world's most popular platform, go to **msdn.microsoft.com/visualj**

Microsoft®
VISUAL
Studio

© 1998 Microsoft Corporation. All rights reserved. Microsoft, Windows, *Where do you want to go today?*, Visual J++, Visual Studio, and the Visual Studio logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

ANOTHER 10 MILLION TRANSACTIONS TODAY. WITHOUT A HITCH.

Looking for excitement? Not from your mission-critical applications. You want to wake up every morning knowing these applications are up and running as expected. Which is why the world's largest companies depend on BEA's middleware.

From transaction processing monitors to object transaction managers to Web and Java application servers, we give networked applications the mission-critical performance, scalability, and reliability they require. We provide the most proven end-to-end middleware solutions, along with the professional services and support to ensure your success.

Find out more about BEA, carrying billions of valuable transactions around the world every day, without a hitch. Call us at 1-800-817-4BEA, or visit www.beasys.com.

BEA

PROVEN MIDDLEWARE SOLUTIONS. (BECAUSE YOUR JOB IS EXCITING ENOUGH.)

February 1999
Vol. 4, No. 2

Java™ REPORT

THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT • www.javareport.com

Technical Features

ENTERPRISE JAVA

The Java International API: Beyond JDK 1.1 27

Mark Davis and Helena Shih

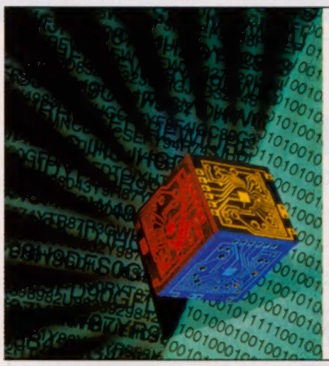
The internationalization services in Java 1.1 provide a wide range of functionality and are easily extended to add additional features and to support additional countries. Discussed are design decisions taken in developing these classes and some of the enhancements that will be included in future releases. How to find out more about the JDK i18n classes and future internationalization improvements that are underway between IBM and Sun.

VISUAL JAVA

Using Resource Bundles to Internationalize Text 37

Bob Patten and Garry Grandlienard

Internet applications are, by definition, graphical in nature. As a result, you end up with a large number of strings due to text on buttons, messages, dialogs, labels, lists, and so forth. Most applications need to quickly switch between languages for this text in order to support the Internet's worldwide community. This results in having to support multiple versions of an application and adds a burden on the developer unless forethought is given on how to handle this large number of strings. Java supplies several ways in which these strings can be handled.



POWER JAVA

Efficient Text Searching in Java 45

Laura Werner

Text searching and sorting is one of the most well researched areas in computer science. It is covered in introductory algorithms courses in many engineering schools. Laura provides ideas on how to use collators to add language-sensitive sorting and searching to your own Java applications. The benefits can be enormous because global markets are becoming more important to the computer industry. Using the international features of the JDK can help you begin to tap into this huge market.

JAVA PRIMER

Object-Oriented Software Development in Java 59

Todd Lauinger

The Java language, its APIs, and recent Java development environments present a powerful toolset for software development. Todd describes the most widely used techniques in object-oriented software development today. These techniques should ensure that whatever code is produced will first and foremost satisfy user requirements and second, be designed to stand the test of time.

Java™ REPORT
THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT • www.javareport.com
February 1999 Vol. 4, No. 2

Columns

Effective Java 62
Supercharged Strings
Steve Ball
Described are the immutability of strings, which has a number of happy consequences for the Java developer. Read about how to take advantage of special properties that improve the efficiency of code that works with string objects. Also, find out about what the Java Language Specification has to say about supercharging string operations and why not everything it says is what it seems.

JavaScripting 70
Client-Side processing?
Steven W. Disbrow
One of the main reasons JavaScript exists is to take some of the load off of the server and put it on the client. This means one thing: data validation. JavaScript is used to validate the data in a HTML form and that data is then sent to the server where some process uses it to generate a result page that is sent back to the client.

Durable APIs 74
Designing durable APIs in Java
Mark Davis
Large-scale system design is different than designing for a smaller program. Mark reviews how to design durable APIs that can withstand the test of time.

Scott's Solutions 80
Printing in Java
Scott Oaks
Scott explores issues surrounding printing in Java. He reviews answers to a variety of related questions, from the basics of how to print something in Java to more complex issues of printing things that span pages.

Departments, Reviews, News

Editor's Note 6
Looking for JDK 1.2?
Dwight Deugo

Letters to the Editor 8

Java Buzz 10
Philip J. Gill
Just How Big Is EAI?; Active Software Delivers Third-Generation EAI; No Linkage Between Sun, DOJ Suits, Said Analyst; RSA Expands Java Security Line; and more.

Coffee Talk 14
The Glue That Holds Diverse Systems Together
An interview with Symantec's Mansour Safai
John K. Waters
From Café's initial release back in 1996, Safai has been there shepherding an evolving line of development tools. Find out what this "insider" predicts for Java's future.

Product Review 17
Java XML Parsers
Juancarlo Añez
A comparative evaluation of 7 free tools.

Ad Index 64

Check out Java Report Online!

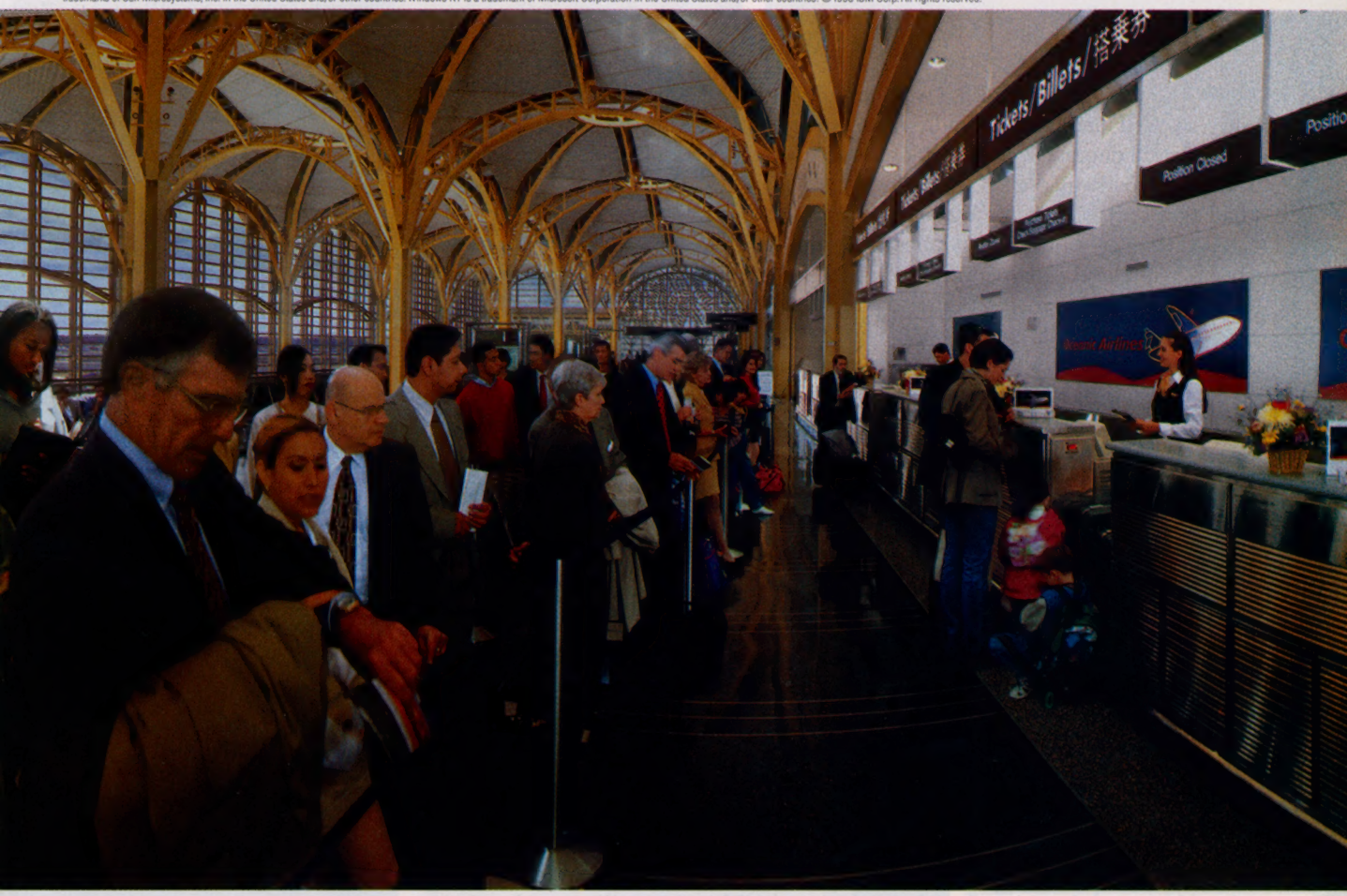
- *Selected features and columns from the print edition*
- *All 1998 issue code listings are available online!*
- *Monthly product updates and product reviews*
- *Coming in 1999: Java Report Archives*

Cover image by Steven Hunt/Image Bank

e-business

FREE Web Self-Service Starter Pack includes DB2® Universal Database™ trial code, tools and customer examples. www.software.ibm.com/websoftware.

IBM, DB2, DB2 Universal Database, AIX, the e-business logo and Solutions for a small planet are trademarks of International Business Machines Corporation in the United States and/or other countries. Sun, Solaris, Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries. Windows NT is a trademark of Microsoft Corporation in the United States and/or other countries. ©1998 IBM Corp. All rights reserved.



Lose the wait. Put service online with DB2 Universal Database,

the database designed for e-business. Fully Java™-enabled, it serves up text, graphics, sound and video to bring self-service apps to life on the Web. All with "right now" performance for thousands of simultaneous users – so everyone's first in line. On the back end, built-in connectors make it easy to consolidate data from virtually any source (including competitive databases) on any platform. Speaking of which, DB2 Universal Database itself runs natively on a world of platforms: from Windows NT® to Sun™ Solaris® and AIX®. The trial code is free at www.software.ibm.com/websoftware. Where you just might discover your company's first beneficiary of Web Self-Service is you.

Solutions for a small planet™ IBM

Looking for JDK 1.2?

Dwight Deugo
deugo@scs.carleton.ca

THIS PAST DECEMBER, I went to New York for the Java Business Expo and, like all media types, made sure to attend Sun's 1.2 launch event. The event was a choreographed, professionally staged performance, as good as any of the off-Broadway plays I've seen. And, like Java 1.2, the event started late. As I had asked myself before, was Java 1.2 going to be worth the wait?

I don't enjoy waiting around much, preferring to spend any free-time I have on the side of a mountain or on a wall at a local rock climbing gym—yes that's really me in the picture. Sometimes, I manage to find the hold I need. Other times I push and pull my way higher, praying that there is something a little further on I can use. In many respects, the Java community has been doing the same thing: waiting, looking, and praying.

As it turns out, we will have to wait forever for Java 1.2. Why? The technology formerly known as JDK 1.2 is now called the Java 2 Platform—I can just imagine the problems this will cause for Sun's documentation version control system. The good news is that Sun announced the availability of the Java 2 Platform on Dec. 8, 1998. So, call it JDK 1.2, the Java 2 Platform, the next release, or just Java, it's ready.

The message from Sun is clear, in fact I am tired of hearing it, "The Java 2 Platform is stable, complete, secure, accessible, and FAST." The point of the message is that Sun believes the Java 2 Platform is ready for prime-time, and that it's up to you and me to build and install network-centric software applications enabling Java to be all that it can be. I hope you're up to the challenge. Sun has done their part.

In hindsight, the name change is a good idea. This is a major release from Sun and any major release deserves its own name. Call it what you want, I am just happy that Sun has continued to emphasize speed. The Java 2 Platform is faster than its 1.1.x counterpart and although the HotSpot technology did not make it into this release, all indicators point to it being there soon. Remember, you also have a part in improving your application's performance. For example, if you try to instantiate an entire `Company` object from a relational database, including all the employees, addresses, contact information, and so on across the network, you can't blame Java for poor performance. Blame yourself.

In this issue, we look at some of the internationalization services provided by

Java™ Report (ISSN 1086-4660) is published monthly by SIGS Publications, 71 West 23rd St., 3rd Floor, New York, NY 10010, 212.242.7447. © Copyright 1999 by SIGS Publications, Inc. All rights reserved. Periodicals postage paid at New York, NY, and additional mailing offices. Regular subscription rates— $39 domestic, $64 Canada and Mexico, $88 foreign; institutional—$139 domestic, $164 Canada and Mexico, $188 foreign. Postmaster: Send address changes and subscription orders to: Java Report, P.O. Box 5050, Brentwood, TN 37024-5050. GST (Canada) #R895884229. For service on current subscriptions: in North America call 800.361.1279 (v) or 615.370.4845 (f); (custserv@cwixmail.com (email); outside North America +44(0)1858 435 302 (v), +44(0) 1858 434 958 (f); custserv@cwixmail.com (email). Statements of opinion and fact are made on the responsibility of the authors alone and do not imply an opinion on the part of SIGS Publications, Inc. or the editorial staff. All trademarks are the property of their respective owners. Printed in the USA.

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other countries. *Java™ Report* is independent of Sun Microsystems, Inc.

Please send unsolicited manuscripts to Managing Editor, *Java™ Report*, 71 West 23rd St., New York, NY 10010, email: kmajor@sigs.com.

Java™ REPORT
THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT

Editor-in-Chief
Dwight Deugo; *deugo@scs.carleton.ca*
Senior Managing Editor
Kathleen M. Major; *kmajor@sigs.com*
Senior Art Director
Andrea Cammarata; *acammarata@sigs.com*
Assistant Managing Editor/ Associate Online Editor
Dan Olawski; *dolawski@sigs.com*
Assistant Art Director
Fernando Quiñones; *fquinones@sigs.com*
Editorial Assistant
Sherese Robinson-Lee; *srobinson@sigs.com*
Associate Editor
Philip J. Gill; *philipgill@aol.com*
Contributing Editor
John K. Waters; *john@watersworks.com*
Product Editor
Kevin Weeks; *kweeks@tidalwave.net*
Product News
Jane Grau; *jgrau@hudsonet.com*
Editorial Intern
Rachel Allison; *rallison@sigs.com*

COLUMNISTS
Steve Ball, *Independent Consultant*
Mark Davis, *IBM*
Steven Disbrow, *EGO Systems*
David S. Frankel, *Genesis Development Corp.*
David Geary, *Independent Consultant*
Vincent Hardy, *Sun Microsystems*
David Jordan, *Ericsson*
Lowell Kaplan, *Random Walk Computing*
Craig Larman, *Object Space*
John Miller Crawford, *Dave Clark Design Associates*
Scott Oaks, *Sun Microsystems*
Tig Tillinghast, *Internet Profiles*

SIGS PUBLICATIONS, INC.
Richard P. Friedman, *Founder, President, & CEO*
Margherita R. Monck, *Vice President Finance/Circulation*
Gayle C. Ehrenman, *Executive Editor*

SIGS ADVISORY BOARD
François Bancilhon, *Ardent Technology*
Grady Booch, *Rational Software Corp.*
Alan Brown, *Sterling Software*
Stuart Frost, *SELECT Software Tools*
Thomas Keffer, *Rogue Wave Software*
Bertrand Meyer, *ISE*
Meilir Page-Jones, *Wayland Systems*
Jan Popkin, *Popkin Software and Systems, Inc.*
Kenneth S. Rubin, *Innolution*
Dave Thomas, *Object Technology International*

SIGS PUBLICATIONS

Publishers of *JOOP, Component Strategies, C++ Report, Java™ Report, ObjektSpektrum (Germany), Java™Spektrum (Germany), Application Development Advisor (UK), Java™ Report Online, Component Strategies Online.*

Zero G Software's

InstallAnywhere™ 2.5

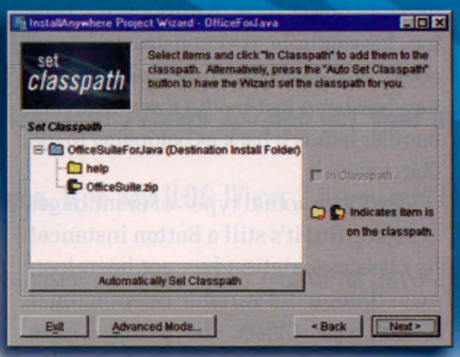
New Features Include:

- SpeedFolders™ - faster-than-native installations
- Java 2 support
- Optimized for large installations
- Install with or without a Java™ VM
- LaunchAnywhere™ applications launchers for: Windows 95, 98, NT, Mac OS, OS/2, Solaris, Linux, HP-UX, IRIX and AIX

Professional Look and Feel

Installers have a professional look and feel that's familiar to all users on all platforms. You can run installers from CD-ROM or directly from the Web.

Create Installers Quickly

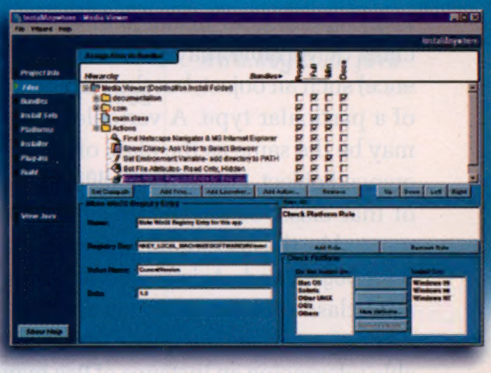

InstallAnywhere Project Wizard - OfficeForJava
set classpath
Select items and click "In Classpath" to add them to the classpath. Alternatively, press the "Auto Set Classpath" button to have the Wizard set the classpath for you.
Set Classpath
OfficeSuiteForJava (Destination Install Folder)
help
OfficeSuite.zip
In Classpath
Indicates item is on the classpath.
Automatically Set Classpath
Exit
Advanced Mode...
< Back
Next >

In six steps you can create an installer with the Project Wizard. It sets the classpath and finds "main" classes automatically.

JDJ Java Developer's Journal — JDJ Editor's Choice Award

Editors' Choice — JavaWorld 1998 WINNER

Power and Control



Need more power and control? The Advanced Designer lets you control everything! Create custom install sets, icons, installation rules, multiple destinations and more...

InstallAnywhere Standard Edition $995
InstallAnywhere Express Edition $495

New Optional Add-ons:
- Internationalization East
- Internationalization West
- Advanced Actions Pack

zero SOFTWARE

FREE TRIAL VERSION available at www.ZeroG.com

Zero G Software 118 King Street, Suite 415 San Francisco, CA 94107-1916 Phone: (415) 512-7771 Fax: (415) 512-7775

the Java 2 Platform. Because the many network applications today involve doing business around the world, these services should be on the mind of every developer. Related to the internationalization theme, I just hope at next year's Java Business Expo, the registration software is willing to accept my Canadian address, but I don't think it's up to the Java 2 Platform to solve that problem!

Dwight Deugo

LETTERS TO THE EDITOR

SEMANTICS OF TYPE VS. CLASS

I enjoyed reading the Effective Java column ("Rescuing instanceof from abuse," Vol. 3, No. 10) in the October, 1998 issue. I believe it's important to present tutorials on object-oriented basics because many current Java programmers have no previous experience with object-oriented languages. Hence, a column on polymorphism in Java is welcomed.

However, if you intend to instruct novices, it's important that the information you present is correct and complete (the whole truth and nothing but the truth). In particular, the article's content involves the semantics of type vs. class without fully explaining the differences. There's a fundamental tension between these two concepts in Java because it is simultaneously strongly typed and class-based.

An object is an instance of one specific class: When you create an object, you typically say something like new Dog(), creating an instance of the Dog class. A variable may point to (reference) such an object, but the variable is of a particular type. A variable's type may be the same as the class of its referenced object, any of the superclasses of that class, or any interface implemented by the class or its superclasses. So if Dog extends Animal, and Animal is a subclass of Object, and Animal implements the Animate interface, then a variable referencing an instance of Dog may be of type Dog, Animal, Object, or Animate. The object referenced by the variable is still an instance of the Dog class.

It's important to understand there is a difference between a variable's type and an object's class. When we send a message to an object that is referenced by a variable, the method that gets invoked depends on the receiver object's class, not the variable's type. The method will always be one defined in, or inherited by, that class. Just to confuse things, when a message includes arguments, the current types of those arguments (not their classes!) helps to determine which method is invoked (this is particularly important when the method name is overloaded); nonetheless, the method choice is still dependent on the object's class.

Returning to the column, the following code appears on page 63:

```
Public boolean action(Event event, Object
 what) {
   if (event.target instanceof Button)
       System.out.print("Button: ");
   if (event.target instanceof Choice)
       System.out.print("Choice: ");
       ...
```

The authors add: "There is no alternative but to use instanceof here, because within action(), the actual type of event.target has been lost behind its ostensible Object type. Although the button, for example, was constructed as an object of class Button, that type information has been lost by the time the event handler hands it back as belonging to the Object class."

Yes, "the actual type" of event.target is Object. But it's still a Button instance! Its class association has not been lost, even though the variable referencing it is of type Object. In general, the event.target object is still an instance of Button, or Choice, or whatever. This is precisely why all the instanceof operators still work. Further, regardless of the type of the thing returned by event.target, it is an object and we can still send messages to it. So an alternative to this code does exist:

```
Public boolean action(Event event, Object
 what) {
   System.out.print(
event.target.getClass().getName() + ": ");
   }
```

Actually, this code will result in printing "java.awt.Button: " and so on. But you see the point. First, when we send messages to an object referenced by a variable, it is the object's class—not the variable's type—that determines the response. Secondly, an object never "forgets" its class pedigree, regardless of the type of variable referring to that object.

Sherman R. Alpert
alpert@watson.ibm.com

JAVA NETWORKING

I read the article, "Java Networking," by Ronald Crawford II in the November (Vol. 3, No. 11) issue and thought you may be able to recommend some Java classes that I may be able to use. I am trying to copy files between two UNIX machines. Is it possible to accomplish this from within Java? Thanks for your help.

Albert Rodriques
rodriqua@bms.com

Ronald Crawford's Reply

Please elaborate a little bit about the totality of the problem that you are trying to overcome. Are you trying to copy files between UNIX machines with Java or are you trying to copy files in general? This will help me help you resolve your problem rather quickly (we hope). Thanks for reading the article in the Java Report *and look for other forthcoming articles! So, what did you think of the article?*

Ronald@Crawford.net

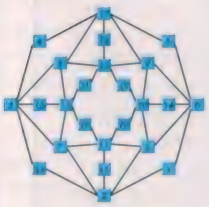
RODRIQUES' FOLLOW-UP

My problem is this: There is a Java application running on a UNIX machine A. Its function (among other things) is to copy some files from machine A to a UNIX machine B. There is no JVM on machine B.

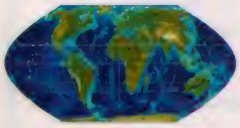
I am planning to write a native method that makes a call to a C function which then makes a call to UNIX to execute a shell script that FTPs the files over. Is this the best way. I was hoping there was an easy way to do this from within Java (without having to write client and server socket code). I notice that with the URLConnection class and a data stream I could open the file on machine B and copy them to machine A, but not the other way.

TAKE THE BRAKES OFF YOUR JAVA GUI

Create super animated icons faster than ever before



Simplify complex data relationships



Add "smart" web-based mapping

Drive ILOG JViews 2.0 and get your applications out of the slow lane. With new ILOG JViews 2.0, you won't be limited to the simplistic forms-based interface that's typical for Java apps. You'll speed into a whole new world of highly interactive, visually rich GUIs that you can build quickly and easily. Your application can now include intelligent, web-based maps, relationship graphs, and detailed, data-aware icons. And like the award-winning ILOG JViews, version 2.0 delivers industry-leading speed and performance.

For new applications or web-enabled versions of existing ones, nothing comes close to ILOG JViews 2.0. Take a free test drive. Visit our website at **www.ilog.com/java**.

ILOG is a registered trademark of ILOG. ILOG JViews is a trademark of ILOG. All trademarks, service marks and company names are the property of their respective owners.

ILOG
Tel: 1-800-FOR-ILOG
jviews@ilog.com
www.ilog.com/java

ILOG

The world leader in software components

ZeroG Supports Multi-Platform Deployment

Zero G Software Corp. (www.ZeroG.com) of San Francisco has come out with a new version of its multi-platform installer tool, InstallAnywhere 2.5 Standard Edition. Available as an upgrade to the existing InstallAnywhere 2.0, InstallAnywhere 2.5 includes new features that speed performance and aid multi-platform deployment of Java applets and applications. The product's new SpeedFolders technology provides for the fastest installation possible, the company claims. It also surpasses the performance of platform-specific native installers, they said.

For the first time, InstallAnywhere 2.5 support multi-platform deployment on Windows 95, 98, NT, UNIX, MacOS, OS/2, and all other Java-enabled operating systems. Another unique feature of InstallAnywhere 2.5 is the option to include a Java virtual machine (VM). When the Java VM is included, InstallAnywhere is the only Java-based installer that can automatically install Java as part of the installation eliminating the need to download and install Java separately.

Alternately, installers can be built without including the Java VM, thereby making the installers smaller and faster to download. In this mode, installers run on the pre-installed Java VMs included with today's operating systems. InstallAnywhere is fully compatible with JDK 1.2, allowing developers to deploy their software for use with the latest VM releases from **Sun Microsystems Inc. (www.sun.com).** InstallAnywhere 2.5 Standard Edition is available now for US $995. The upgrade price for current 2.0 Standard Edition users is US $395.

Just How Big Is EAI?

The market for EAI Middleware products is a fast-growing opportunity for Java vendors. Indeed products such as **Active Software Inc.**'s **(www.activesw.com)** new ActiveWorks 3.0 software are vying for leadership in the market. So how big is EAI and how fast is it growing? The folks at Active Software have assembled the following statistics from some of the leading market research houses. The different market research houses define the EAI market a little differently, and some use a slightly different nomenclature, but they all mean essentially the same thing. And they all see fast growth.

For instance, the folks at **Gartner Group Inc. (www.gartner.com)** of Stamford, CT, expect the EAI market to increase at 45% annually to become a $6 billion market by the year 2000. The **Yankee Group Inc. (www.yankeegroup.com)** of Boston, predicts what it calls the application integration software market to reach about $5 billion by the end of year 2001, also growing at an annual rate of 45%. The **Aberdeen Group Inc. (www.aberdeen.com)** estimates that EAI will be $1 billion-plus market over the next 2 years, equaling or exceeding the market for traditional Middleware. And finally, **International Data Corp. (www.idcresearch.com)** of Framingham, MA, said the Application Integration Engine segment, its name for EAI software of the market outpaced the growth of the broader Middleware market with growth at a rate of 85.1% during 1997.

Active Software Delivers Third-Generation EAI

Active Software Inc. (www.activesw.com) has come out with the third generation of its Java-based Enterprise Application Integration (EAI) Middleware. Now called ActiveWorks 3.0, the new release offers as its main enhancement what the Santa Clara, CA, company claims as "unlimited scalability" to integrate diverse applications across multiple computing systems, departments, organizations, and geographies. It integrates legacy, packaged, custom, Internet, and database applications. Using its new Multi-Broker option, companies can configure an unlimited number of Active Software's Information Brokers (which link disparate applications together) across multiple systems, departments, even separate organizations, to provide event-driven integration of business-critical systems. Previously, Active Software's Information Broker was limited to a single instance.

Other new features include: fail-over protection; internationalization; a new Administrative API for developing custom programs to monitor and control ActiveWorks; and expanded platform support. With this release, for instance, ActiveWorks 3.0 now supports **Hewlett-Packard Co.**'s **(www.hp.com)** HP-UX UNIX operating system, in addition to Solaris and AIX UNIX, and Windows NT.

by Philip J. Gill javabuzz@sigs.com

No Linkage Between Sun, DOJ Suits, Said Analyst

Could **Sun Microsystems Inc.**'s **(www.sun.com)** first-round victory in its legal battle with **Microsoft Corp. (www.microsoft.com)** presage a similar victory for the **Department of Justice (DOJ; www.doj.gov)** and its anti-trust suit against the Redmond, WA, software giant? Not likely, said a senior analyst at **Summit Strategies Inc. (www.summitstrategies.com)**, a Boston-based IT market strategy research firm.

Dwight Davis, Services Director of Summit Strategies' Microsoft as Partner and Competitor Service, has urged caution in linking the preliminary injunction issued against Microsoft in its Java licensing dispute with Sun with the broader anti-trust action underway against Microsoft in the Federal courts. The ruling in the Sun case orders Microsoft to modify any of its products that include Java code so the code will pass Sun's tests for Java compliance.

"People shouldn't extrapolate too far from the two very narrow issues the Java ruling addresses," said Davis. "Only one of these very technical issues has potential relevance to the Department of Justice's anti-trust case in our opinion."

Davis said he finds it ironic that, in part, the Java ruling requires Microsoft to support the Java Native Interface (JNI) going forward. "The JNI is intended to give developers a way to have their Java-language programs directly access the services of a specific underlying operating system in this case, Windows," said Davis. "Applications that take advantage of the operating-system-specific JNI code, by definition, won't automatically run unaltered on other operating systems. In other words, these applications fall outside of the much-heralded "write once, run anywhere" capability that has always been Java's most-compelling promise."

Davis said he believes the second aspect of the preliminary injunction is more troubling for Microsoft, particularly in light of the DOJ's anti-trust suit. "It suggests that Microsoft introduced new code into its Java implementations and labeled it 'Java' code even though it wasn't part of the official Java specification," he said. "This action has been characterized as a transparent attempt to pollute the "pure" Java specification, which would undermine Java's "write once, run anywhere" functionality even for some developers who thought they were developing fully portable Java applications. If this part of the ruling holds, it will reflect very badly on Microsoft's business practices, and could well end up as an important element of the federal anti-trust case."

RSA Expands Java Security Line

RSA Data Security Inc. (www.rsa.com) has expanded its line of Java security products in an effort to position itself as a "one-stop shop" for Java security components." The San Mateo, CA-based company has added an SSL product, dubbed BSAFE SSL-J, and renamed its existing Java security product, now called BSAFE Crypto-J. The new BSAFE SSL-J 2.0 component suite is intended to give developers everything they need in a single package to implement the Secure Socket Layer (SSL) v3 protocol in Java for applications in banking, financial services, Web publishing, consumer, and electronic commerce. It includes extensive sample code, API libraries, certificate authority components, a full suite of socket libraries, and full user and API documentation. The enhanced BSAFE Crypto-J 2.0 product "formerly known as JSAFE" continues to implement

all the core RSA cryptographic algorithms used in major application security standards, such as SSL, Secure Electronic Transaction (SET), and Secure Multipurpose Internet Mail Extension (S/MIME). It also adds support for Java Crypto Extensions (JCE) and the Digital Signature Algorithm (DSA).

Software developer kits for BSAFE SSL-J and BSAFE Crypto-J are available now for $295. Runtime licenses for products and support using BSAFE SSL-J 2.0 or BSAFE Crypto-J are available on a royalty, per-seat, or annual fee basis.

Cloudscape Details Java Database Releases

Cloudscape Inc. (www.cloudscape.com) has detailed the next two versions of its Cloudscape all-Java embedded database for mobile applications. The first new version, Cloudscape version 1.5, is available now, while the next release, Cloudscape version 2.0, is currently in beta with companies such as **Crossroads Technologies, Endpoint!,** and **SAIC.**

The chief enhancement in Cloudscape 1.5 is the concept of the Virtual Table Interface (VTI), which the company said allows Java programmers to integrate data from a variety of sources, thus, making all data sources appear to an application as if they were tables in a Cloudscape database. Version 1.5 also includes single- or multi-user license options, and optional packaged integration with the BEA WebLogic all-Java application server, from **BEA Systems Inc. (www.beasys.com)**, for turnkey application development.

Also included are performance enhancements and a cost-based optimizer to enhance query performance through automated system optimization. Cloudscape 1.5 has a 1.5 MB footprint and supports JDBC 2.0 and ODBC 3.0. Currently in beta, Cloudscape version 2.0 includes an advanced synchronization technology, called LUCID, short for Logic Up, Consistent Information Down. LUCID provides exceptional conflict resolution, without requiring custom coding at remote sites. This ensures the user that there is application consistency from operational applications to deployed applications. It will also support row-level locking for higher multi-user concurrency in deployed server environments. Cloudscape 2.0 should be available in production release in the second quarter of this year.

The Second Coming of Sun's picoJava Chips?

With a little help from some licensees, **Sun Microsystems Inc. (www.sun.com)** hopes to breathe new life back into its family of dedicated Java microprocessors for the embedded systems market, the picoJava family. At the Comdex/Fall 1998 show, four Sun picoJava licensees disclosed their plans for employing the dedicated Java chips in upcoming products. They include **ETRI, Fujitsu Ltd.'s Fujitsu Microelectronics Inc.** subsidiary, **NEC Corp.**, and **Siemens AG**.

The Electronics and Telecommunications Research Institute (ETRI; **crown.etri.re.kr**), a Korean government institute charged with developing information and telecommunication technology, showed off NetTheater, an Internet set-top box reference design it developed using a picoJava-based chip and all Java software. The Java-enabled set-top box, the first product to embed microJava processor from **LG Semicon (www.lge.co.kr)**, allows for execution of the Java applets and applications through the Web. Besides supporting basic functions such as Web surfing and sending and receiving email, NetTheater is the Java-enabled Internet Set-top box supporting multimedia applications on the PersonalJava 1.1 environment.

Fujitsu Ltd.'s **(www.fujitsu.com.jp)** microelectronics group disclosed that its Java solutions, using Fujitsu's advanced LSI technologies, will be targeted at digital consumers and the emerging networked personal computing and communications market, including smart handheld devices, digital still communications markets, and car navigational systems. Through its semiconductor group, Fujitsu is offering Java system LSI solutions, including the picoJava core for high-speed execution of Java byte codes. Fujitsu also revealed that its JavaOS all-Java operating system evaluation board J-StarterKit for SPARClite with Java OS on SPARClite is available now. Fujitsu is a second-source and OEM of Sun's SPARC RISC CPU architecture, including the SPARClite chip for portable computers and other low-end devices. The J-StarterKit for picoJava evaluation board for the picoJava chip is expected to be introduced this year.

The NEC Semiconductor Group of NEC Corp. **(www.nec.com.jp.)**, for its part, said its Java technology emphasizes as a controller to many embedded devices such as home appliances, telecommunications systems, factory floor controllers, and vending machines. NEC is also focusing its product development on reducing power consumption and creating smaller cores that maintain good Java performance. NEC believes that the picoJava core is well suited for use as an IP core and said that it will focus on using the picoJava core as an IP core in its "System on a Chip" (SOC) concept, which has enabled adding high functionality to controllers in embedded devices. NEC said it has also developed an evaluation chip of the picoJava core and has run it on an evaluation board. By using the evaluation board, NEC said it will continue to develop the software environment, including an RTOS kernel and Java runtime.

Siemens AG's **(www.siemens.de)** semiconductor operations disclosed its plans for developing the high-end controller 88-series of smart cards. Siemens' 88-series smart card chips accelerate the execution of the Java Card Instruction Set while addressing the needs of multi-application cards. Siemens said it also plans to develop chips that will be used in a variety of smart card applications, including banking, electronic commerce, mobile communications, and healthcare. In addition, Sun demonstrated, for the first time, JavaOS and JDK 1.1.4 running on the microJava 701 silicon. Based on the picoJava-II core, the microJava 701 is a high-performance, general-purpose microprocessor developed to validate the picoJava architecture.

SevenMountains Brings TaskForce to U.S.

A Scandinavian software company has set up a shop in Silicon Valley and launched what it hopes will become a market-leading, all-Java desktop and collaborative software product. The company, called **SevenMountains Software Inc. (www.sevenmountains.com)** has opened offices in San Mateo, CA, to bring its TaskForce product to the attention of U.S. customers and OEMs.

The company describes TaskForce as "the first fully integrated enterprise desktop and productivity suite written entirely in Java." The product is designed to be a framework for building enterprise-class, team-oriented, business applications in server-side Java components. Potential applications include Human Resources, Project Management, and other team-based tasks within an enterprise environment.

TaskForce, based on a multi-tier architecture, comprises the Silverstream application server from **Silverstream Software Corp. (www.silverstream.com)**, a unified desktop, and a suite of collaborative productivity tools. All are written in 100% Java. Built on the TaskForce Framework, the basic system includes email; calendar; document creation, publishing, and management; full-text retrieval; a conference system; a Web browser; a 3270 emulation for access to legacy applications; an interactive organizational chart, and an office suite, including a word processor, spreadsheet, presentation, Gantt diagrams for project management, address books, and business graphics.

Moreover, the TaskForce Framework is easily expandable and customizable, as all applications are viewed as Java components. To add new functionality, a developer need only add in a new off-the-shelf or custom-built Java component. For this reason, the company expects ISVs and enterprise IT organizations to be its main audience.

The company said it has TaskForce in beta at a limited number of enterprise customer sites, though it's too early to release names. However, it expects to have some customers available to discuss their use and development of TaskForce-based applications by the end of the first quarter of this year.

AOL's Netscape Deal to Boost Java, JINI

Java and JINI might well be big winners in **America Online Inc.**'s **(AOL; www.aol.com)** buyout of **Netscape Communications Corp. (www.netscape.com)**, which is being aided by Java and JINI's developer, **Sun Microsystems Inc. (www.sun.com)**. AOL has agreed to acquire Netscape for about $4 billion—with a little help from Sun. Under their definitive three-year development and marketing agreements, AOL and Sun will work together to develop a suite of easy-to-deploy, end-to-end solutions to help companies and Internet service providers rapidly enter the e-commerce market and scale their e-commerce operations. Sun will become a lead systems and service provider to AOL, which, in turn, has agreed to purchase systems and services worth $500 million at list price from Sun through 2002 for its e-commerce partners and its own use.

AOL will also receive more than $350 million in licensing, marketing, and advertising fees from Sun, plus significant minimum revenue commitments, during the next three years. A key part of the AOL/Sun alliance is a commitment by Sun and AOL to develop the next generation Netscape Navigator and Communicator Web browser software clients. In addition, AOL will support the Java technology from Sun in its e-commerce solutions, including the Java 2 Platform as well as PersonalJava.

A major focus of the alliance will be the use of PersonalJava to offer AOL services across a range of next-generation Internet devices. This is consistent with AOL's "AOL Anywhere" strategy to extend its brand to all emerging mass-market platforms. PersonalJava is specifically designed for small, nomadic devices such as a personal digital assistant (PDA), cell phone, or pager. Thus, the AOL/Sun agreement will further establish the Java platform as a leading software environment for these emerging Internet-based personal and business tools.

javabuzz art: p6, Campbell Laird/SIS; p7, Kamalova/SIS.

Oracle JDeveloper Suite. The

all-terrain

Java development tool.

On the client. On the application server. On the database.

You'd expect that applications written in Java could run anywhere, wouldn't you? After all, that is the big promise of Java. The problem is that most Java applications don't end up pure. Well, things are about to change, because now there's Oracle® JDeveloper Suite. A set of tools and servers optimized for building real world database applications and components using 100% pure, unadulterated Java. They'll be able to run on any platform, or connect to non-Java and other legacy applications. No other Java development tool lets you leverage the full potential of Java across the client, application server and database. There is no out-of-bounds. To find out more and get a free demo CD of Oracle JDeveloper Suite, visit *www.oracle.com/info/27* or call 1-800-633-0539, ext. 19146.

And find out what a world without borders really means.

ORACLE®

©1998 Oracle Corporation. All rights reserved. Oracle is a registered trademark of Oracle Corporation. HUMMER® and the vehicle's grille design are registered trademarks of AM General Corp. HUMMER® image is used courtesy of AM General Corp. All other names may be trademarks of their respective owners.

COFFEE TALK

The Glue That Holds Diverse Systems Together

Java Report interviews Symantec's Mansour Safai

__Mansour Safai__ is vice president and general manager of Symantec Corporation's Internet Tools Division. Under Safai's direction, Symantec released Café, the first IDE for Java, and later, the Visual Café product line, Visual Page, and the dbANYWHERE server.

Before joining Symantec, Safai was president and CEO of MultiScope Inc., a debugging tools company he founded. MultiScope earned a solid reputation in the software-debugging marketplace and was acquired by Symantec in 1992. Safai had worked previously as manager of the Software Tools group at Logitech Inc., where he served as project leader for the MultiScope debugger product. He worked on the Post-Mortem Debugger, the graphical user interface, the graphical representation of data structures, and the port of the software to OS/2. Safai also was responsible of the Modula-2 development team. He worked on the compiler, the code generator and several components of the runtime system.

By most counts, the ranks of Java tool vendors (the ones with products you have to pay for) now number an unsteady 18. Even with all that competition—or perhaps because of it—Symantec Corporation continues to lay claim to approximately 60% of the Java development tools market. Since Symantec released Café, the world's first Java IDE, back in March 1996, its products have won high praise and every computer magazine award of consequence.

From that first release back in 1996, Mansour Safai has been there, shepherding Symantec's evolving line of Java development tools. Safai lives on the inside track, not only of his company's products, but also of Java itself. Beyond the Internet, Safai sees Java technology emerging as the "glue" holding an increasing variety of diverse systems together now and in the future.

Safai talked with our Silicon Valley correspondent, John K. Waters, from Symantec's Cupertino, CA, offices.

Java Report: **Lots happening at Symantec these days. Feel like flogging your products a bit?**

Mansour Safai: Always! Though we don't really have to "flog" our product line. We make great tools that speak for themselves.

Spoken like a true believer.

I am that.

Okay, late last year, Symantec announced the release of Visual Café 3.0. The newest version of your venerable devtool carried with it a new emphasis on server-side development. Why?

We've been in the Java tools business since the very beginning. We provided the very first Java development tool, called Café. The announcement was about the tool's shift in the direction of developing a Java toolset dedicated to developing enterprise-class applications. It tries to keep development very similar to the desktop development. The whole product line is called Visual Café Enterprise Suite.

You've been watching the Java show virtually from the beginning. How are things stacking up in terms of the evolution of Java technology?

Originally when Java showed up, a lot of people got very excited about the idea of being able to write only one time for one platform and being able to deploy on a number of platforms afterwards. In practice, people realized that Windows had already won the desktop war. There really was no need for a language that would allow you to run on multiple platforms from a desktop point of view.

With Windows in ascendance on the desktop, why hasn't Java just dried up and blown away?

Because people started paying attention to something that was stronger and more important than the desktop: gluing enterprise systems together that have been written on multiple platforms and in different languages—gluing them together and allowing people to take advantage of their newly found Intranet- or Extranet-type infrastructures to develop business applications.

You're talking about the fairly recent, some might say, wholesale shift of Java development to the server side.

> ***"When Java showed up, a lot of people got very excited about the idea of being able to write only one time for one platform and being able to deploy on a number of platforms afterwards."***
>
> **—Mansour Safai**

THEY SHIP IT WITH A PRIZE IN EVERY BOX.

Inside every box of the world's most popular development suite is the power of the world's leading visual modeling technology.

"By working together to closely integrate modeling and development tools, Microsoft and Rational have made modeling more accessible to all Windows developers. Microsoft Visual Modeler, developed from Rational Rose technology, is now included in every box of Visual Studio. This partnership provides a great way for developers to take advantage of the full functionality which Visual Studio provides."

Paul Gross
Vice President Development Tools Division, Microsoft

RATIONAL SOFTWARE

www.rational.com/mvs/vismod/
1-800-728-1212

Microsoft
Enterprise
Development Partner

Rational, the Rational logo, and Rational Rose are trademarks or registered trademarks of Rational Software Corporation in the United States and in other countries. Visual Studio 6.0 is a trademark or registered trademark of Microsoft Corporation. ©1998 Rational Software Corporation. All rights reserved.

That's correct. I remember two years ago everybody was talking about Java in the browser. That's really what got it going. But what seems to be way more important is Java in the middle tier, Java in the database, Java for transactional systems, Java as the glue between different systems. On the server side, Java seems to be doing a very good job.

Why?

What made it really hard for Java to be truly cross-platform on the desktop is the graphical user interfaces (GUIs). It's hard to get them to be the same. But on the server side, there's not a whole lot of UI going on. Most of the work is in the area of algorithms and logic. If you think about it this way, that part of Java is very highly portable. You would have very little problem running Java on several servers, especially if there's no user interface, which is typical of server-side applications.

So you're saying that server-side Java is thriving because it doesn't have to deal with GUIs?

That's a big part of it. The other thing is memory management requirements. On the desktop, you launch an application, use it, and then exit the application, which typically has the effect of cleaning up all the memory that was utilized by that application. But servers are running week after week after week. At some point, the objects that were created in the memory of the machine have to be purged or reallocated so that new objects can be created or at some point you're going to run out of memory.

Okay, so Java has automatic garbage collection and good memory management. Is that why it's better suited for server-side development?

It's better and it isn't.

Um...thanks for clearing that up.

[laughs] It does have automatic garbage collection and pretty good memory management altogether, as compared to languages that are just C or C++. On the other hand, if you talk to customers they will tell you that, depending on the type of applications they have, they want a very specific type of memory management.

For example?

Some of them don't want the garbage collector to step in every few minutes and try to collect memory that is not being utilized on the system. It would slow down the server too much. They want that to kick in maybe every 12 hours. You do enter a whole new area of requirements by customers.

> ***"Java 2 has to be very highly stable and it must perform."***
> **—MANSOUR SAFAI**

You must be seeing developers moving into this space or your tools would not be evolving in this direction. Am I correct in assuming that people have been asking for the features you're delivering with Visual Café 3.0?

[The evolution is] based on what I call our "unfair advantage." We have now more than 100,000 users of our Java products and we've been able to get a lot of data from them that has really helped us to understand what's happening in the marketplace and where they want to go with Java and why. The direction we have taken is definitely based on customer demand, not where we wanted to go.

They're telling you that they're developing complicated distributed applications?

They are, and they want us to make it easy. A developer should really be concentrating on his logic, the piece of the application that he's writing. He's not an RPC or CORBA expert. He is a database developer, a user interface developer, or a business applications developer. We really spent a long time with people to try and understand this, and we realized that it was a lot harder to do it this way, but we did it because it allows customers to choose best of breed. That's really what enterprise customers want.

I've heard it said that we're at the start of a new Java era. What do you think about that statement?

It's an era that's going to be very, very different from the one we've seen in past. I think it's one that is a lot closer to the glass house. I think it's one in which you are going to have to work with legacy machines and apps and backends.

What are your customers saying about the future of Java?

They're saying, listen, Java was new a couple of years ago, now we're really trying to do some serious development here. The tools have to become much more reliable, polished, and stable. We are in the adoption cycle of Java tools. Maybe a year ago you could go out there with the latest and coolest features. Now you really have to go out there with something very solid.

I have a press release here in which you are quoted: "This is the third generation of our Java development tools, and as such are very mature products." What about Java itself? Do you see these changes—the demands of your customers, the shift toward server-side development, the evolution of your product line—as a sign of a maturing of Java technology?

Yes, definitely. Everybody is getting very serious.

How would you describe your experience working with Sun?

From a tool vendor point of view, they've been a good partner. We've received the latest versions of the JDK early, we got the changes early, we've been involved in the specifications of things like the new EJB spec and new APIs. Those things, we are the first ones to know. Again, we have our "unfair advantage:" it takes me about 25 seconds to go from my office to the JavaSoft door.

Any thoughts on the Sun/Microsoft lawsuit?

Not specifically, but I will say this: Personally, I have a lot of respect for Microsoft. I wake up in the middle of the night worrying about how to compete with them. They're good, and that keeps me sharp.

What's missing from Java right now?

Java 2 has to be very highly stable and it must perform. If those two things are dealt with, we don't need additional

continued on page 26

Java XML Parsers

A Comparative Evaluation of 7 Free Tools

Reviewed by Juancarlo Añez

RECENTLY, MODELISTICA HAS been conducting an evaluation and feasibility study to determine the suitability of XML and Java for the representation and manipulation of Transport and Land Use (TLU) modeling information as used in urban and regional planning.

XML, the Extended Markup Language, is highly publicized as the replacement for HTML for describing document content on the Web. But markup languages have a long history and have applications far beyond those of the Web. They are currently being used for information description and exchange in such diverse areas as finance and trade, mathematics, chemistry, biology, knowledge representation, genealogy, software package description and distribution, CASE, graphics, and more.

MARKING UP

HTML is the most popular and most widely known use of markup. XML was designed by the World Wide Web Consortium—often referred to as W3C—to enable the use of the Standard Generalized Markup Language (SGML) on the Web. XML is a public standard: it is not a proprietary development of any single company. The version 1.0 specification was accepted by the W3C as a formal Recommendation on Feb. 10, 1998. The XML Web page at the W3C site is the entry point to a sea of information about XML, SGML, and related technologies and applications.

XML is an abbreviated version of SGML—the international standard for defining the structure and content of electronic documents. XML eliminates the more complex and unused features of SGML making it much simpler to implement, but still compatible with its ancestor. XML is actually not a single language but a meta-language. XML can describe both the syntax of specific classes of documents, and their contents. The portion of XML that determines document syntax is named the Document Type Definition language (DTD). XML supports multiple DTDs.

From the XML perspective, HTML is just one of these document types—the one most frequently used on the Web. It defines a single, fixed type of document with markups that let you describe a common class of simple office-style reports. Because it provides only one way of describing information, HTML is overburdened with dozens of interesting but often incompatible inventions from different manufacturers. In contrast, XML allows the creation of markup languages customized to the needs of specific applications—which is what brought me to investigate the possibilities of defining a markup language for urban planning. If you're interested, a long list of current and under-development applications of SGML and XML can be found at OASIS' XML Web page.

Table 1. From version 0.4 1 to version 1.9, the claimed capabilities all over the map.

PRODUCT	VERSION	JAR SIZE (in KB)	JDK 1.2 SUPPORT	VALIDATING	DOM LEVEL 1 SUPPORT	SAX SUPPORT	NAMESPACES	XLINK\ XPOINTER
AElfred	1.2a	32	Yes	No	No	Yes	No	No
DCXML	Beta 1	144	Yes	Yes	Yes	No	Yes	No
XML4J	1.1.4	460	Yes	Yes	Yes	Yes	Yes	Yes
MSXML	1.9	101	No	Yes	No	No	Yes	No
Sun XML	EarlyAccess 1	104	Yes	Yes	Yes	Yes	Yes	No
SXP	0.72	196	Yes	Yes	Yes	Yes	Yes	Yes
XP	0.4	173	Yes	No	No	Yes	No	No

DEFINING THE APPLES AND ORANGES

My first experiments showed that XML files for our area of interest, Transport and Land Use (TLU) information would be large. One to four megabytes looks normal. So it was important that parsers had good performance in terms of both speed and memory usage. Defining an XML document type for TLU is among the project's long-term goals, so attention was also given to the ability to parse XML Document Type Definitions, and validate XML documents against it. I also considered implementation of current and upcoming XML standards.

All the parsers and tools reviewed here are available on Web. As of this writing, no commercial parsers were available and most parsers were flagged with some label indicating the publisher wasn't claiming the parsers were production-quality. These products (with the exception of Microsoft's original parser) are freely available for download. They are releases in all but name.

CONFORMANCE

The XML standard classifies documents into one of three categories: not well formed, well formed but invalid, and valid. A document is well formed when it meets all the syntactic and semantic requirements described in the XML standard. A well formed XML document is also valid when no Document Type Definition (DTD) is provided. When a DTD is provided, a valid document must also comply with the grammar described by the DTD.

Furthermore, documents can be stand-alone or can have references to external information, and the XML standard allows for special treatment of external definitions by non validating parsers.

I used James Clark's XMLTest test suite to evaluate how well the parsers conformed to the XML definition. The XMLTest suite is composed of several hundred small XML files and DTDs, each one testing for conformance with a specific aspect of the XML standard. The tests range from simple checks to highly contrived entity definitions and expansions. The test suite also includes normalized versions of all the valid files so that they can be compared with the output of the targeted parsers.

For validating parsers, I added yet another test. I introduced a simple but obvious error in the first lines of one of the large files I used in the performance test. The name of one of the elements was changed to one that did not appear in the DTD and was, hence, invalid. This was a trivial test and only one of the validating parsers failed it.

NAMESPACES, XLINK, AND XPOINTER

XML namespaces provide a simple method for qualifying names used in XML documents by associating them with namespaces identified by URI. Namespaces are intended to avoid problems of recognition and collision in documents with fragments of different types. An example is the case of a small database described in XML Data, embedded in an HTML document.

The XML Linking Language (XLink) consists of constructs that may be inserted into XML documents to describe links between objects. XLink can describe the simple unidirectional hyperlinks of today's HTML as well as more sophisticated multi-ended and typed links. The XML Pointer Language (XPointer) allows hyperlinks that reference arbitrary document fragments.

The Namespaces, XLink, and XPointer specifications are currently at the "working draft" level, so they were not included in the evaluation. These technologies are important, so you'll find mention of the parsers that implement the current draft versions of the standards.

DOM LEVEL 1

The Document Object Model (DOM) is a language-neutral API that allows programs to dynamically access and update the content, structure and style of documents. The DOM Level 1 Specification is already a publicly available W3C Recommendation.

The DOM defines a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. A specific library can support the DOM as an interface to proprietary data structures and APIs. Applications that use the standard DOM interfaces rather than product-specific APIs, become independent of particular implementations. The DOM standard currently defines language bindings to Java, Corba IDL, and ECMA Script (the European JavaScript/JScript standard).

To evaluate the DOM compliance of the libraries, I wrote a small program to test the interfaces defined in the DOM Java binding as they appear in the `org.w3c.dom` package (see Listing 1). The program was run using each library in turn.

SAX

The Simple API for XML (SAX), is a standard interface for event-based XML parsing, developed collaboratively by the members of the XML-DEV mailing list (see the Microstar Web site). A SAX-compliant XML parser reports parsing events to the application through callbacks, without necessarily building any internal structures. The application implements handlers to deal with the different events, much like it's done by modern graphical user interfaces (GUIs), like Java AWT.

The SAX API makes the parser layer totally independent from other application or library functionality. A particular set of event handlers may be used to build an in-memory representation of an XML document, while a different set of handlers may render the document on the fly. Java packages that implement a SAX driver are in fact interchangeable, at least in theory.

PERFORMANCE

Speed and memory usage tests using two large XML files were performed (0.8 and 1.2 MB, respectively) by one of our in-house applications. Each file contains several thousand XML elements nested in a four-level deep hierarchy, and all of the elements have one or more attributes.

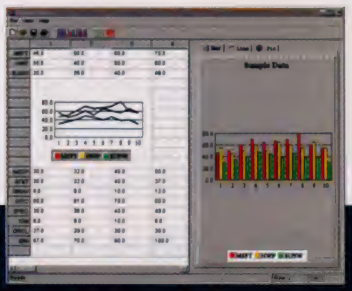
StudioJ

a suite of Java tools

North American pricing
$995
Special upgrade pricing available for existing Rogue Wave and Stingray customers!

- Full source code included!
- No royalties or run-time fees!
- Now with improved JFC support and documentation

Integrated components
for developing great Java™ applications!



Why shop all over for Java components from different vendors and then spend days making them work together? **StudioJ** is your one-stop solution! Now you can get the high-quality, integrated, pure JavaBeans components and classes you need all in one place. To order or upgrade today call 1-800-487-3217 or visit:

www.roguewave.com/ad/studioj

Blend.J for adding over 45 AWT and JFC unique and rich extensions to your GUI front end.

Grid.J for table or spreadsheet layouts, with support for Excel-like formulas, binding to any data source, and much more.

Chart.J for dynamic, customizable charts, with data filters and overlays for composite charts.

DBTools.J for SQL portability across databases, with thread-safe and extensible classes.

Rogue Wave is a registered trademark, and StudioJ and .Jar trademarks of Rogue Wave Software. Java and all Java related trademarks are registered trademarks of Sun Microsystems, Inc. All other trademarks are the property of Rogue Wave Software or their respective holders.

Rogue Wave SOFTWARE

Rogue Wave: 800-487-3217 · Stingray: 800-924-4223 · D.A.C.H.: +49-6103-59 34-0 · UK: +44-118-988 0224
France: +33-1-5568 1008 · Italy: +39-02-3809 3288 · Rest of Europe: +31-20-301 26 26

For each parser and file, three runs were performed: one without validation, one providing the DTD and enabling validation, and a third run using the same scheme as in the second but introducing a validity error in the first 10 lines of the XML file. The DTD is the same for both files. It consists of 530 lines and uses DTD entity definitions (sort of a DTD macro) moderately.

A separate test was performed on the parsers that provide an Object Model to measure model navigation speed, and memory use. The test consisted of loading a large XML file and querying the object model while constructing yet another application-specific structure. To force navigation of the complete structure, the model's Document object (or its equivalent) was used to write a new XML file out to disk. Because this was a performance and not a compatibility test, changes were made to the test program so it would run with those parsers that didn't implement the DOM Level 1 standard, implemented it incompletely, or had their own proprietary object model. Libraries that would have required non-trivial changes were not tested. This test was performed with validation turned off to minimize parser overhead and focus on object model navigation.

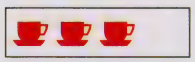
All tests where done using SUN's Java Runtime Environment (JRE) version 1.1.7A on an 300 MHz Intel Pentium II with 128 MB of RAM running Windows 98. The maximum heap space for all tests was set to 64 MB. The programs were compiled using the same version of the SUN javac compiler, with optimizations turned on. Times were measured using an external command line program called from a batch file, so they include the time needed to load the Java VM and any required libraries. Memory usage was obtained by examining the trace of the programs after running them with verbose garbage collection turned on. Note that these tests were not devised as benchmarks that would help determine the split-second fastest parser, nor byte consumption per XML element. They were designed with the intention of exposing problems in parser design that had an obvious impact in performance when working with large XML documents.

DOCUVERSE DOM SDK

The DOM SDK, by Docuverse, is not an XML parser, but a DOM implementation that works on top of any parser that exposes a SAX interface. It is discussed here because the very first tests already showed that it is indeed very simple to combine the DOM SDK with different parsers. Performing the DOM test using the SDK with both the Ælfred and XP parsers shows that these combinations are serious competitors to integrated parsers like Sun's or IBM's.

The DOM SDK is available at the Docuverse DOM SDK page . The license allows free distribution of the binaries (.class and .jar files) but is very restrictive about copying or modification of the source code and documentation.

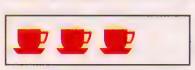
JAMES CLARK'S XP V0.4



XP is a "high performance" XML parser produced by James Clark, who was technical lead for the W3C SGML activity group. This group produced the first draft of the new XML standard. XP is non-validating, but it checks if documents are well-formed, and are capable of parsing external entities including DTDs. The only interface XP provides for applications is a SAX driver, so it qualifies as a lightweight parser. The documentation provided with the parser consists only of the output from JavaDoc. The documentation was too succinct at times and assumes familiarity with SAX.

XP performed in the top tier, along with the parsers from Microstar, IBM, and Microsoft. XP performed well also when combined with the DOM SDK using the DOM test suite. Under this test suite, XP and Ælfred, another lightweight parser, produced almost equivalent results. It is expected these two parsers will evolve in different directions in the near future. James Clark's XP emphasizes conformance, and will probably evolve into a validating parser, while AElfred emphasizes efficiency, portability, and fault tolerance, and will probably evolve in that direction without adding the complexity of new features. XP performed well under the XML conformance test, which is not surprising. After all, James Clark himself devised the test suite. The XP parser is free and is available at James Clark's Web site.

MICROSTAR ÆLFRED V1.1



Ælfred is a parser that concentrates on optimizing speed and size rather than error reporting. This approach is

Listing 1.

A fragment of the code used to test for Document Object Model Level1 compliance.

```
import org.w3c.dom.*;
import com.modelistica.TRANUS.model.*;

//...
static Model makeModel(Element top) {
    //...
    // NODES
    section = (Element) top.getElementsByTagName("nodes").item(0);
    items = (NodeList) section.getElementsByTagName("node");

    for(int i = 0; i < items.getLength(); i++) {
        Element e = (Element) items.item(i);
        com.modelistica.TRANUS.model.Node n =
            model.newNode(e.getAttribute("id"));
        n.setX(attToFloat(e.getAttribute("x")));
        n.setY(attToFloat(e.getAttribute("y")));
        addProps(n, e.getChildNodes());
    }
    top.removeChild(section);
    //...
}
```

We know the territory. We understand the latest Intel® processors and know what each one can do. So for programs that fully utilize the Intel® Architecture, trust the people who make it in the first place. Intel's VTune™ Analyzer is critical in improving all of your hard work. The VTune Analyzer provides a system-wide view of all of your code and how it runs on Intel processors. Whether you're creating programs that run in C/C++, FORTRAN, or JAVA, they can execute a lot quicker. It's even faster than ever at pinpointing bottlenecks and eliminating any roadblocks, buddy.

VTune™ Performance Enhancement Environment

Download a free 14-day evaluation at: http://developer.intel.com/vtune

intel

©1999 Intel Corporation. All rights reserved. Intel is a registered trademark and VTune is a trademark of Intel Corporation. All other brands and trade names are the property of their respective companies. Performance comparisons are between VTune Analyzer versions 3.0 and 2.5.

the most useful for deployment over the Internet. Ælfred consists of only two core class files, the main parser class (XmlParser.class) and a small interface for your own program to implement (XmlProcessor.class). All other classes in the distribution are either optional or demonstrations. At 31 K, Ælfred's JAR file was, by far, the smallest among all the parsers.

Ælfred uses only JDK 1.0.2 features, but testing showed that it runs fine with JDK 1.1.6, 1.1.7A, and 1.2rc1. The documentation claims that the parser is compatible with most character encodings available on the Internet, but no attempt was made to test that assertion.

This parser was designed to be very lightweight, very portable, and very fault tolerant. It will produce correct output for well-formed and valid documents, but it won't necessarily reject every document that is not valid or not well formed. Ælfred will probably never become a validating parser.

Ælfred comes with very complete API documentation in the form of HTML files generated by JavaDoc 1.1. Several simple example projects are also included. This parser was fast in the tests that didn't involve validation, and was able to complete the DOM test when combined with the Docuverse DOM SDK. Ælfred and XP performed almost equally.

The conformance test showed that Ælfred is not as fault tolerant as the documentation suggests. Ælfred generated exceptions for valid documents that were not stand-alone, and went into an endless loop of error reporting for some of them. Ælfred failed to report many documents that weren't well formed.

Ælfred is free for both commercial and non-commercial use and redistribution. The only requirement is that Microstar's copyrights are preserved in derivative source code, and that any modifications are clearly documented. Ælfred can be downloaded from Microstar's site.

MICROSOFT XML (MSXML) V 1.9

MSXML is a validating XML parser produced by Microsoft as part of its Internet Explorer 5 effort. The parser has support for namespaces and is compliant with the XML draft specification of November 1997. The parser provides its own Object Model, which is quite powerful but isn't DOM Level 1-compliant. MSXML does not provide a SAX driver, but drivers are available elsewhere—check out Lars Marius Garshol's Free XML Software page and the Microstar Web site.

MSXML's documentation consists of several sample projects and JavaDoc documentation for the API. The API documentation is nicely laid out, but many of the methods are undocumented in this version. The sample projects include some interesting ones like an XML viewer applet. Another set of applets can take small databases described in XML Data and lay them out nicely using tables and dynamic HTML. Some of the applets even allow for the edition of the XML Data information, from changing field values, to adding and deleting records.

MSXML was the top performer in terms of both speed, and memory usage. The parser performed better than the small SAX driven parsers in all tests, despite the fact that MSXML always builds an in-memory model of the document and validation was always turned on. In the DOM test, MSXML consumed only half the memory of its closest rival. All this performance fits in a JAR file of just 101 K, which gives the parser the smallest footprint among those that provide an object model. Also note that MSXML's performance is provided through 100% Pure Java code. Whatever the secret is to MSXML's performance, other parsers would do well imitating it.

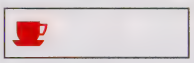
The DOM test had to be adapted to be run with MSXML. The algorithm remained the same, but many declarations and method calls had to be changed. MSXML performed quite well on this test. It's speed and memory performance was better than that of any of the other parsers. The API is not DOM-compliant, but it is as expressive as DOM, so it shouldn't be difficult to make MSXML DOM Level 1 compatible.

On the conformance test, MSXML gave incorrect warnings and errors about many valid documents. The parser also failed to detect many of the documents that were not well formed or invalid. MSXML does not provide a SAX driver, but drivers are available on the Web (as mentioned previously).

MSXML originally didn't work with SUN's JDK 1.1.6 or 1.1.7A, because two locations in the library's initialization code assumed that the JDK version would be convertible to a float value. The Integrated Development Environment (IDE) used to construct the tests suites promptly pointed me to the faulty lines, so I fixed them. Oddly, MSXML reported an invalid document with JDK 1.2 on a test that ran to completion with JDK 1.1.7A.

Microsoft entered into an agreement with Data Channel for further development of the parser. At this writing, MSXML had been removed from the Microsoft Web site. Unfortunately, the current beta of the parser provided by Data Channel is evaluated below MSXML in all regards. Fortunately, the license Microsoft provided with its version 1.9 parser is liberal enough that you'll likely be able to find copies of the original, or of its heirs elsewhere.

DATA CHANNEL XML PARSER



The XML parser from Data Channel (DCXML) is derived from Microsoft's. Surprisingly, the package layout and the methods available in the DC parser are very different from those in MSXML. DCXML performed well below most other parsers in all tests for speed and memory use. Even though DCXML is on its first beta, no differences with the base code layout and performance were expected.

The documentation provided with DCXML consists of the output of JavaDoc over a set of Java files with absolutely no JavaDoc comments. As such, the documentation is useful for browsing through the source code and little more.

DCXML performed well below the other parsers in all tests in terms of

Extending your business to the Web has never been faster or easier, thanks to new Progress Apptivity 3.0,™ a complete e-business solution. With Apptivity 3.0, you get the industry's most comprehensive, integrated, and scalable development environment and application server. And you get a host of new features. Like SmartClient, which adapts to the capabilities of your browser. And SmartConnect, which keeps your applications in touch with all your information, including databases, packaged applications, and legacy systems. Progress Apptivity 3.0 is fully standards-based, supporting JavaBeans, Enterprise JavaBeans, CORBA, HTML, XML, JDBC, and more. It's also designed to be extremely easy to use for rapid deployment. Gain your competitive advantage today. Call 1 800-477-6473, ext. 4700 or download Progress Apptivity 3.0 at www.apptivity.progress.com/rocket

PROGRESS
SOFTWARE

© Copyright 1999 Progress Software Corporation. All rights reserved. Progress, the Progress Software Logo and Apptivity are all trademarks of the Progress Software Corporation.

PROGRESS APPTIVITY 3.0 — EXTENDING YOUR BUSINESS TO THE WEB

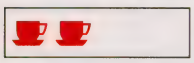
speed, but it was able to complete tests that Sun XML couldn't when the Sun parser ran out of memory.

Object model tests on DCXML were not performed because it lacked the equivalent of the DOM method `getElementsByTagName()`. I could perform that test on MSXML because it provides the same functionality through an `Element.getChildren().item()` method.

In the conformance test, DCXML failed to recognize about 15% of valid documents, generating null pointer exceptions for several of them. DCXML had only a few problems with documents that were not well formed. Most of the errors occurred in documents that had references to external entities.

The licensing policy for DCXML is currently unknown. The license that was bundled with the downloaded parser is an exact copy of the liberal one that came with MSXML 1.9. A different version of the license in Data Channel's Web site states that the parser is free for commercial use as long as some value added is provided. Yet another version of the licensing policy was received via email, stating that the parser was free only for non-commercial use. This parser is in an early beta state, and its characteristics and the related policies may change considerably by the time it's released.

SUN XML, EARLY ACCESS 1



The Sun XML Library consists of a fast parser with optional validation. It has a SAX interface and the library provides an object model that is DOM Level 1 compliant. Sun's XML Library is labeled "Early Access 1", which means it's still under construction.

The parser's API documentation was generated by JavaDoc 1.2 and it's very complete. SUN also provides several sample programs that highlight library features such as DOM, namespace support, and JavaBean support. The set of sample programs serve well as a tutorial about the libraries' capabilities.

As in other libraries built around the SAX API, the parser and the object model are completely independent. SAX compatibility enables you to use the Sun parser core with other applications, including other DOM implementations like Docuverse's DOM SDK. The class in charge of building the in-memory object model, the `DocumentBuilder` class, implement's the SAX `DocumentHandler` interface, which enables the use of Sun's object model with other SAX-compliant parsers, like XP.

Sun's parser performed quite well in the tests that did not involve validation. On the tests where we included a DTD to provide validation, times were comparable to those of the fastest parsers but memory consumption skyrocketed. With validation enabled, the parser failed with an "out of memory" exception and was not able to complete the test with the 1.2 MB XML file. The test that involved a file with an invalid element on the first few lines consumed as much memory as when the file was parsed entirely.

The parser also performed very poorly on the DOM navigation test, taking more than 25 minutes to complete. The results made Sun's DOM implementation the worst performer. On the conformance test, Sun XML failed to recognize just one of the valid documents.

Current licensing for the Sun XML parser is restricted to evaluation purposes. The licensing policy for the finished parser has not been disclosed by Sun. The parser can be found at the Java Developer's Connection page.

IBM XML4J (XML FOR JAVA) V 1.1.4

XML4J is a validating XML parser produced by the IBM alphaWorks project. The parser is compliant with the XML 1.0 standard, it has support for namespaces and DTD manipulation and it implements DOM Level 1. XML4J also provides a SAX driver. XLink/XPointer support are also provided but were not tested.

The documentation for XML4J consists of a tutorial, complete API documentation in HTML and several sample projects. The tutorial covers all relevant features of the library with special attention to the libraries unique features. The API documentation was generated by IBM's own implementation of JavaDoc. The API descriptions are very complete and the HTML layout has a quality comparable to that of documentation generated by JavaDoc 1.2.

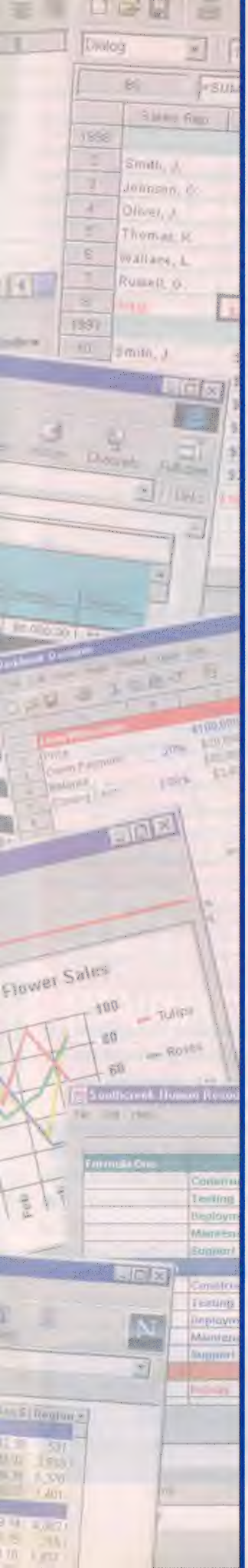
One of XML4J's unique features is the possibility of setting up event handlers and filters at the object model level. This feature enables you to build an object model out of only selected parts of a complex XML document. But XML4J's functionality comes at a price. The JAR file that contains IBM's library is 460 K, which is four times the size of the libraries from Sun and Microsoft, and more than twice the size of the libraries for DCXML and XP.

XML4J was among the top performers in the lot, falling only behind the MSXML parser overall, and behind the lightweight parsers only in the tests that did not involve validation. IBM's parser worked fine on the DOM test. Memory use was elevated when compared to that of MSXML and the lightweight parsers on the simple tests but was well below that of other parsers in the validation and DOM tests. Unlike Sun's library which behaved like the lightweight parsers when no object model was requested, IBM's parser consumed the same amount of memory when used as a SAX driver and when a DOM model was requested.

This parser went through the XML conformance test with very few problems. It failed to recognize only three documents as not being well formed. XML4J is distributed with full source code and a free commercial license. The parser is available for download at IBM's alphaWorks' Web site.

LORIA SXP V 0.72

SXP is an XML library produced by Loria in France, as part of their ambitious XSilfide project. SXP is implemented as a SAX driver and has support for DOM Level 1, namespaces, XLink, and XPointer. The documentation that comes with the library consists of JavaDoc generated HTML pages, but the comments are few and succinct. The documentation was un-



Spreadsheet needs come in all sizes.

Fortunately, so does our technology.

Formula One Professional is the most versatile spreadsheet solution available when you need to deploy spreadsheet functionality.

Java developers can use it as a JavaBean in development environments such as Visual Café and JBuilder to implement spreadsheet functionality in their server- or client-side applications.

Webmasters can use it to add live spreadsheets to their HTML pages with as few as one line of code.

End users on any desktop can use it as a standalone application.

Use it in any thin-client situation to deliver spreadsheet functionality to numerous users from a server.

Even use it to connect to databases and read and write Excel files on any platform or through the Internet.

A complete solution, Formula One Professional also delivers its spreadsheet functionality as an ActiveX component, enabling it to be implemented in Windows applications built with development environments such as Visual Basic.

Formula One™ Professional

- ✓ Formula One for Java® 5.5.1
  - ✓ Application, applet, and JavaBean
  - ✓ Read/write Excel files
  - ✓ 100% Pure Java
- ✓ Formula One ActiveX™ 6.0
  - ✓ Read/write Excel files
  - ✓ First Impression ActiveX 6.0 charts
- ✓ Fast calculations
- ✓ Excel formula and function syntax
- ✓ Workbook designer
- ✓ Robust API
- ✓ Database connectivity
- ✓ Small footprint and downloads
- ✓ Server and client side functionality
- ✓ Write spreadsheets to HTML
- ✓ Per user and server/CPU pricing

So the next time you need robust spreadsheet functionality, make the same choice businesses such as State Street Bank and Simon & Schuster have already made.

Turn to **Formula One Professional**, industry-leading technology for delivering spreadsheets everywhere!

Tidestone TECHNOLOGIES™

Technology for delivering SPREADSHEETS EVERYWHERE™

activeX

100% PURE JAVA

JDJ EDITOR'S CHOICE AWARD

INFORMATION: **888-880-0665** • ***DEMOS AND FREE FULL-FEATURED TRIALS:*** **www.tidestone.com/sheets**

Copyright © 1999 Tidestone Technologies, Inc. All rights reserved. Tidestone, the Tidestone logo, Visual Components, Formula One, First Impression, and VisualSpeller are trademarks of Tidestone Technologies, Inc. Java, 100% Pure Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. ActiveX and Excel are trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners. Specifications subject to change without notice.

helpful in trying to discover the unique characteristics of the library.

SXP's performance was poor in terms both of speed and memory use. SXP was notably slow in the tests that involved validation, taking at least 15 minutes to complete any of them. But poor performance is to be expected of a parser in beta—optimizations are best left until the end of the development process. On the plus side, the DOM test ran unaltered with the SXP library.

On the conformance test, SXP failed to recognize more than 20 documents as valid, and it failed to detect the errors in a like number of documents that were not well formed or were invalid. SXP is free for academic, research, and non-commercial use. The library is available at Loria's Web site.

CONCLUSION

Here, we've reviewed both beta versions and officially shipping products. In the case of MSXML we even looked at a retracted product. Ordinarily, a comparative review such as this would be patently unfair to all products involved. But this is a unique situation. All products are free and all are available on the Web. Whatever games the producers may be playing with labels like "beta" or "early access" or "version 0.72", the products have all been released.

That said, nothing can be concluded about their current performance or about their limitations, if any. As an example, when evaluations began, IBM's XML4J at version 1.0.4 was one of the worst performers in terms of speed, memory use, and correctness. Things changed considerably with the 1.1.4 release and was one of the best parsers we reviewed. These parsers will certainly continue to improve or "die" and do so at breakneck speed. Our intention was to help you evaluate the current crop of parsers.

As the evaluation results show, different parsers excel under different requirements. The lightweight Ælfred from Microstar is the obvious choice when deploying simple applets on the Web. When XML document validation is required IBM's XML4J and Microsoft's MSXML are probably the right choices, with excellent standards compatibility being in favor of the former, and outstanding speed being the latter's claim to fame. For applications like the one we're working on, DOM Level 1 compliance is of primary concern because good compliance with that standard makes the parsers almost plug-and-play, and hence completely replaceable.

Our evaluation made it clear that combining XML and Java is definitely viable, that performance in this combination does not have to be an issue, and that conformance with the standard is rapidly improving.

There are several possible categories. The first one is that of very lightweight and very forgiving parsers like Ælfred, for use on the Web. The second is that of welterweight parsers, that implement most core standards, and provide a reasonable balance between performance, size, and features (I expect Sun's parser to evolve in this direction). Then there's that of heavyweight parsers which implement all relevant standards, are very strict about validation, and provide a wealth of additional features, at the expense of some speed and higher memory requirements. I suspect the lightweight and welterweight parsers will converge on an ideal feature set. The heavyweights will remain and a totally different category will result from the integration of parsers with other types of programs, like Web clients and servers. All the lights are green on XML.

Note: *As this issue went to press, SUN released a new version of its XML parser. Although it was released too late to be included in this review, quick tests showed that speed and memory performance have improved to levels comparable to those of MSXML and XML4J.* ■

Juancarlo Añez is a Consultant with Modelistica, in Caracas, Venezuela, which provides technical and professional services in urban and regional planning. He can be contacted at juancarlo@modelistica.com.

URLs

W3C's XML Page
www.w3c.org/xml

OASIS' XML Web Page
www.oasis-open.org/cover/xml.html#applications

James Clark's XMLTest Suite
www.jclark.com/xml

XML Data
www.w3.org/TR/1998/NOTE-XML-data-0105/

The DOM Level 1 Specification
www.w3.org/TR/1998/PR-DOM-Level-1-19980818/

Microstar Software
www.microstar.com

Docuverse DOM SDK
www.docuverse.com/domsdk/index.html

Lars Marius Garshol's Free XML Software Page
www.stud.ifi.uio.no/~larsga/linker/XMLtools.html

Sun's XML Parser at the Java Developer's Connection Page
developer.javasoft.com/developer/earlyAccess/xml/

IBM's alphaWorks XML Parser Page
www.alphaworks.ibm.com/formula/xml/

Loria's Web Page
www.loria.fr/projects/XSilfide/

COFFEE TALK *continued from page 16*

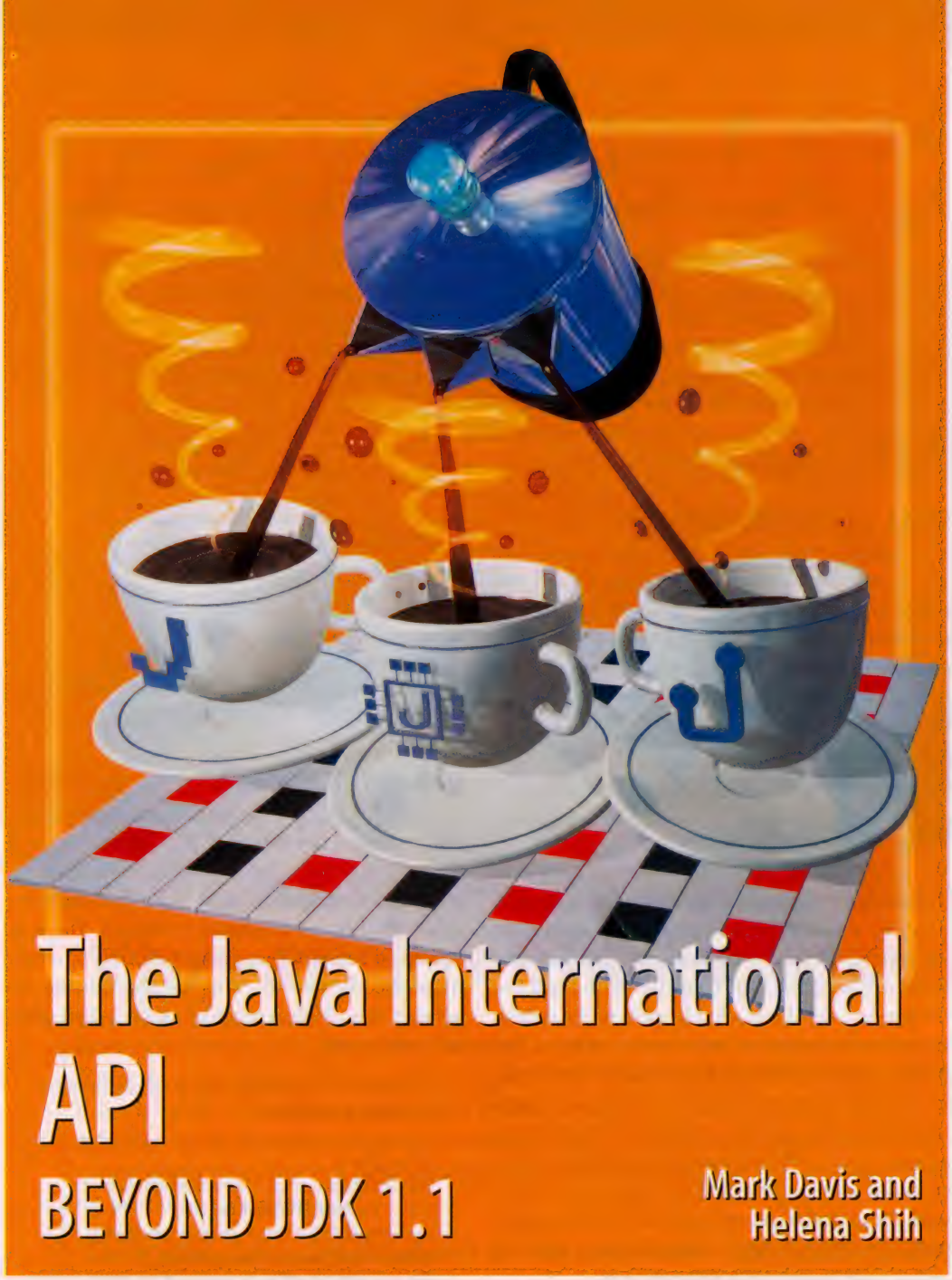
APIs or functionality or God knows whatever service added to it. The current definition of Java, if it was working in a very reliable fashion, would be fine. Things have to look good and run fast. It's that simple. I know it's complicated to get a platform to be highly stable. It took Microsoft many years and many versions to get Windows right.

You could say that. Since Windows 98 was released, with its 3000-plus bug fixes, I've heard Windows 95 described as the biggest beta test in history.

I think it's similar here. It takes time for a technology to mature, to become stable and to perform. I think the customers would be very happy with that. ■

John K. Waters is a Contributing Editor of *Java Report*. He can be contacted at john@watersworks.com.

Coming in March...
Our issue on
Distributed Computing



Roy Wiemann/Image Bank

The Java International API

BEYOND JDK 1.1

Mark Davis and Helena Shih

Important new features under development in mid-level internationalization services

THE DESIGNERS OF Java made the important design decision that all text would be stored in Unicode. This solves the problem inherent in most other text-handling schemes, of always having to juggle multiple, limited character encodings. It puts all languages on an equal footing, and makes the whole process of designing for worldwide products far easier. But proper support of international text requires far more than just storing characters in Unicode.

IBM's wholly-owned subsidiary, Taligent, had a great deal of previous experience in Unicode software internationalization. In 1996, Sun contracted with Taligent to design and develop classes for the proper handling of multilingual text for JDK 1.1. Our goals were to provide an architecture that supplied the required functionality, was fully object-oriented (OO), could easily be extended to add additional features or to support additional countries, and would scale well across both small and large projects written in Java. There are some aspects of the architecture that we frequently get questions or complaints about, so we'll explain why we made some of the decisions we did.

In JDK 1.1 we focused on the "server level" support: that is, on the mid-level internationalization services. Most of the low-level text services were already in JDK 1.0 and only needed

some enhancement. The high-level international services (for input and output) utilized the host platform services in JDK 1.1. In Java 2 the high-level international services have been greatly improved and no longer depend on the host platform services.

We preview some of the most important new features under development in the mid-level internationalization services. Many of these features are in Java 2. IBM is making others available through different channels, including classes available on the IBM alphaWorks Web site. (Note: IBM also has C and C++ versions of its Unicode internationalization services at its Web site.)

DATE AND TIME SUPPORT

The Calendar class contains an API that allows you to interpret a Date according to a local calendar system, even non-Gregorian ones. It also contains routines to support GUI requirements, such as rolling, adding, and subtracting dates and times. The TimeZone class enables the conversion between universal time (UTC) and local time. It also contains rules for figuring out the daylight savings time according to the local conventions.

Why is January Zero?

We probably get more complaints about this than any other issue. Here's what happened. The JDK 1.0 Date API and implementation were very specific to the Gregorian calendar, and were not terribly Y2K friendly. Although the Gregorian calendar is used in most of the world, many countries use other calendar systems. For instance, businesses in Europe often use a calendar that measures by day, week, and year, rather than day, month, and year. There are also a large number of traditional calendars in widespread use in the Middle East and Asia. To deal with these problems, we split out the date computations into another class, Calendar, and retained Date purely as a storage class.

The zero-based month numbers in Date were a vestige of old C-style programming—originally month names were stored in a zero-based array and the months were numbered accordingly for convenience. JavaSoft felt that consistency with the old Date APIs were important, so we needed to keep this convention in Calendar. So calling calendar.set(1998, 3, 5) gives you April 15th, *not* March 15th.

Time Zone Display Names

Every abstract class in the internationalization framework, except for TimeZone, has a getDisplayName() function. This means there is no easy way to get the displayable name for a time zone in JDK 1.1. This has led to some confusion, because some people thought that TimeZone.getID() returned a displayable name, when it actually returns an internal programmatic ID, one that should not be displayed to end users. Moreover, the internal IDs themselves were too short and confusing: "AST" could stand for either "Atlantic Standard Time" or "Alaska Standard Time." To remedy this, the new method getDisplayName() has been added to TimeZone in Java 2, and longer more descriptive internal IDs are available.

```
// In JDK 1.1
TimeZone zone = TimeZone.getTimeZone("EST");
SimpleDateFormat sdf = new SimpleDateFormat(
"z", Locale.English);
fmt.getCalendar().setTimeZone(zone);
String name = format.format(new Date());
// name is "Eastern Standard Time"

TimeZone zone =
TimeZone.getTimeZone(
"America/New_York");
String name = zone.getDisplayName(Locale.ENGLISH);
// name is "Eastern Standard Time"
```

Better Y2K Support

Should "01/01/00" be year 2000 or year 1900? JDK 1.1 used the 80–20 rule. This amounts to adding 1900 to the two-digit year, and if the result was more than 80 years in the past, add another 100 (for the Gregorian calendar). The Java 2 method DateFormat.set2DigitStartDate() provides more specific control. This method sets the exact start of a 100-year range in which 2-digit years are interpreted.

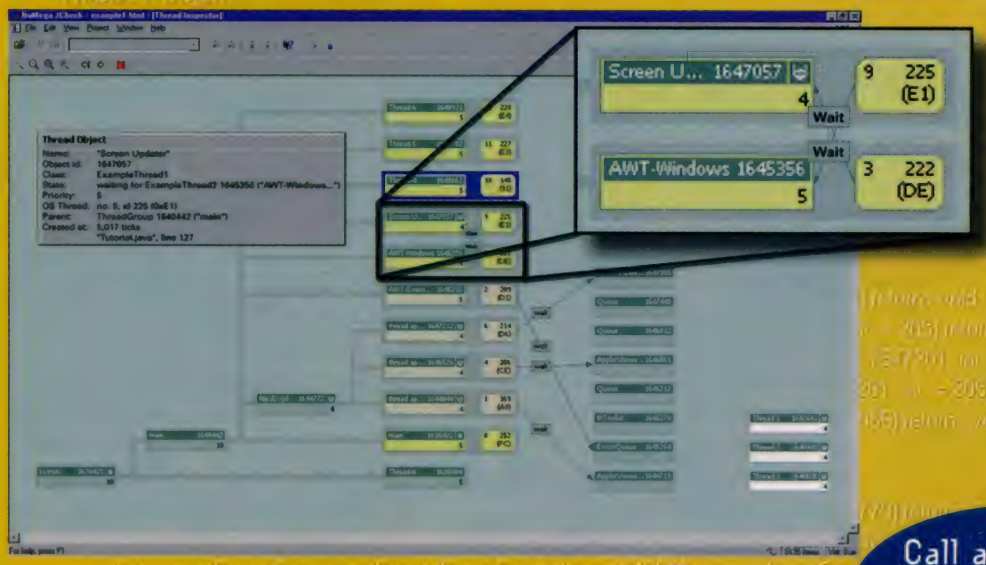
```
// In JDK 1.1: There is no way to specify when the
// 2 digit year starts.

// In Java 2:
GregorianCalendar cal = new GregorianCalendar
(1952, Calendar.SEPTEMBER, 13);
DateFormat fmt = DateFormat.getInstance();
fmt.set2DigitYearStart(cal.getTime());
fmt.parse("9-12-52"); // returns 9-13-1952
fmt.parse("9-14-52"); // returns 9-14-2052
```

Improved Daylight Savings Switchover

In JDK 1.1, SimpleTimeZone allows the start and end dates for Daylight Savings Time to be specified in only one way, as the Nth or Nth-from-last occurrence of a given weekday in a given month, e.g., the last Sunday in October. However, some time zones have more complicated rules for the switchover dates. For example, in Brazil Eastern Time, DST ends on the first Sunday on or after February 11th, which cannot be expressed with the JDK 1.1 APIs.

This was resolved in Java 2 by adding several new types of DST start and end rules. The

DEADLOCK!



Call and request a FREE 14 day license

JCheck™ Exposes Java™ Thread Problems Instantly

Deadlock. Thread Leak. Starvation. Thrashing... Just a few of the problems you'll never have to chase down manually again – if you use NuMega JCheck™, the visual thread and event analyzer for Java. JCheck builds a thread-based graphical model of your program at runtime and provides rich, detailed information on thread states, events, synchronization objects, exceptions, and errors. JCheck expands your runtime visibility in a new dimension, so you will waste less time debugging, and have more time for productive development.

Take your Java development to the next level. Get NuMega JCheck today!

JCheck is also part of NuMega DevPartner™ for Java

Essential tools for Java development

NuMega JCheck™ Visual Thread and Event Analyzer

NuMega TrueTime™ Automatic Performance Analysis and Optimization

NuMega TrueCoverage™ Automatic Code Coverage Analysis

1-888-686-3436

www.numega.com

COMPUWARE
People and software for business applications™

Copyright © 1998 Compuware Corporation. All rights reserved. All trademarks and registered trademarks are the property of their respective owners.

30-Day Money Back Guarantee

following rule types will handle all known modern and historical time zones and provide more flexibility for the future:

1. A fixed date in a given month, e.g., the 1st of April.
2. The first occurrence of a given day of the week on or after a certain date in the month, e.g., the first Sunday on or after February 18th, or equivalently, the first Sunday after the second Thursday.
3. The first occurrence of a given day of the week on or before a certain date in the month.

Correct Rolling

There is no way to implement a date widget with arrow buttons correctly, with the Calendar class in JDK 1.1. Suppose the user has selected the MONTH value for January 30, 1997, and hits the up arrow twice. The first click correctly sets the control to February 28, 1997, but the second click sets it to March 28, 1997, instead of March 30, 1997.

The correct implementation is to remember the original date and roll the month field the proper number of steps from that original date for each click of the arrows. Unfortunately, in JDK 1.1, you can only roll a field a single unit at a time. Java 2 fixes this problem by adding the ability to roll a field, multiple units in a single operation.

```
// In Java 2:
myCalendar.setTime(aDate);
myCalendar.roll(MONTH, numberOfArrowClicks);
```

International Calendar Classes

Although the Calendar class is architected to allow for multiple calendars, both JDK 1.1 and 1.2 only include support for the Gregorian calendar. However, IBM is previewing a large set of international calendars on the alphaWorks Web site—including Hebrew, Islamic, Buddhist, and Japanese calendars.

LOCALES AND RESOURCES

A locale in the JDK is merely an identifier. This identifier is made up of the ISO language code and country code, plus optional variants (for information on the ISO codes, see the Unicode Web site). Because Locale is just a lightweight identifier, there is no need for validity checking when you construct a locale. Whenever you construct an international object, you have the opportunity to supply an explicit Locale or you can use whatever the current default locale is on your system:

```
Collator col = Collator.getInstance(Locale.FRANCE);
if (col.compare(string1, string2) < -1) {
    ... // based on the French locale's sort sequence
```

```
Collator col = Collator.getInstance();
if (col.compare(string1, string2) < -1) {
    ... // based on the default locale's sort sequence
```

The ResourceBundle class provides a way to isolate translatable text or localizable objects from your core source code. For example, resource bundles can be used for translatable error messages, or building translatable components. The JDK also uses resource bundles to hold its own localized data. For example, when you ask for a NumberFormat object, the necessary formatting information is retrieved from a resource bundle.

Why Can't You Set the Default Locale in Applets?

People frequently ask for the ability to call Locale.setDefault() within an applet. The problem is that a single JVM can run more than one applet at a time in the same address space. Locale.setDefault() would change the default locale for the whole address space, which means that all of the applets would be affected; this is considered a security violation. To work around this, set the applet's locale instead of using Locale.setDefault(). When you need an international class, supply the locale explicitly:

```
NumberFormat nf = NumberFormat.getInstance(
    myApplet.getLocale());
```

ResourceBundle Fallback Detection

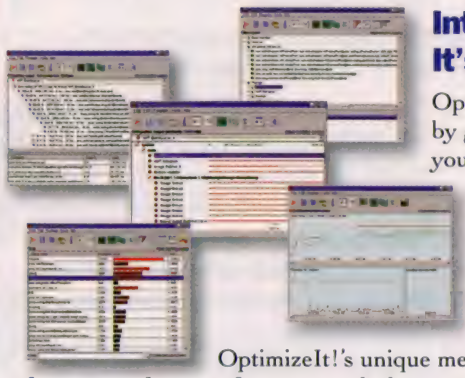
The ResourceBundle implementation currently includes a fallback mechanism: if the specified resource can't be found in the specified locale, ResourceBundle searches:

- first in the resource bundle for the specified language and country
- then in the resource bundle for the specified language
- then in the resource bundle for the default locale's language and country
- then in the resource bundle for the default locale's language
- finally in the root resource bundle

Sometimes this is not what you want, or at least you may want to be able to detect when a particular piece of data came from a fallback locale rather than the specified one. For example, suppose you wanted a specific resource from the French Belgian locale, and there is only a French locale installed—you'll get the wrong resource. In Java 2 we added a method, getLocale(), to find out the actual locale that a resource bundle comes from, so that you can determine if a fallback was used.

Advanced Java™ profiling with OptimizeIt™!

Because you have better things to do with your time than track Java performance problems.



Introducing OptimizeIt! 3.0 Professional. It's Java Profiling Like You've Never Seen.

OptimizeIt! 3.0 Professional makes your Java profiling process easier by giving you the most comprehensive and accurate information on how your Java programs use CPU and memory. What you get is clear, real-time data and easy-to-read graphs allowing you to rapidly pinpoint performance problems such as bottlenecks, excessive object allocations or memory leaks.

OptimizeIt! is the only tool to include a CPU profiler allowing you to tune CPU time measurement to the most efficient level of detail for each test session – while maintaining a low overhead. OptimizeIt!'s unique memory profiler also provides unequaled features such as a reference graph that automatically highlights the references actually causing memory leaks.

But don't take our word for it. Check us out at www.optimizeit.com to download your free trial today, and experience how OptimizeIt! will help you improve the overall performance of your Java programs. And just maybe get you out of the office on time.

Optimize It!

www.optimizeit.com

Optimizeit 3.0 Professional supports JDK 1.1x and JDK 1.2

©1999 Intuitive Systems, Inc. All rights reserved. Optimizeit is a registered trademark of Intuitive Systems, Inc. All registred trademarks belong to their respective manufacturers

```
// In Java 2
Locale frBE_Locale = new Locale("fr", "BE");
ResourceBundle rb =
ResourceBundle.getBundle("MyResources", frBE_Locale);
if (!rb.getLocale().equals(frBE_Locale)) {
        // French Belgian resources not available,
        // report an error
```

COMPARISON AND BOUNDARIES

In JDK 1.1, Collator allows you to compare strings in a language-sensitive way. The standard comparison in String will just do a binary comparison. For strings that will be displayed to the user, this is almost always incorrect! Wherever the ordering or equality of strings is important to the user, such as when presenting an alphabetized list, then use a Collator instead. Otherwise a German, for example, will find that you don't equate two strings that she thinks are equal!

```
if (string1.compareTo(string2) < 0) {...
// bitwise comparison

Collator col = Collator.getInstance();
if (col.equals(string1, string2)) {...
...
if (col.compare(string1, string2) < 0) {...
```

Why Have CharacterIterator?

The CharacterIterator class is used in BreakIterator and a few other places in the JDK and is used even more in Java 2. Sometimes we are asked why we didn't use String or StringBuffer instead.

String and StringBuffer are simple classes that store their characters contiguously. Insertion or deletion of characters in a StringBuffer end up shifting all the characters that follow, which works fine for reasonably small numbers of characters. However, this model doesn't scale well. Consider a word processor, for example, where shifting many kilobytes of characters just to insert or delete one character involves far too much extra work. For acceptable performance in these circumstances, text needs to be stored in data structures that use internally discontiguous chunks of storage.

We needed some way to have a more abstract representation of text that could be used both for String and for larger-scale text models. Unfortunately, we couldn't change String and StringBuffer to descend from an abstract class that would provide this sort of representation. To resolve this problem, we added a very minimal interface, CharacterIterator. This interface allows both sequential (forward and backward) and random access to characters from any source, not just from a String or StringBuffer.

Rule-Based BreakIterator

The BreakIterator class finds character, word, line, and sentence boundaries, which may vary depending on the locale. The JDK 1.1 BreakIterator implementation uses a state machine, which makes it very fast. However, it does not allow the behavior to vary depending on the locale. If the built-in classes don't support behavior the clients want, they must create a completely new BreakIterator subclass of their own—they can't leverage the JDK code at all.

Therefore, we undertook an extensive revision of the BreakIterator framework. The new RuleBasedBreakIterator class essentially works the same way the old class did, but it builds the category and state tables from a textual description, which is essentially a string of regular expressions. This description can be loaded from a resource—allowing different breaking rules for different languages—or supplied by the client—allowing runtime customization. This class is provided on the alphaWorks Web site.

Locale-Sensitive Searching

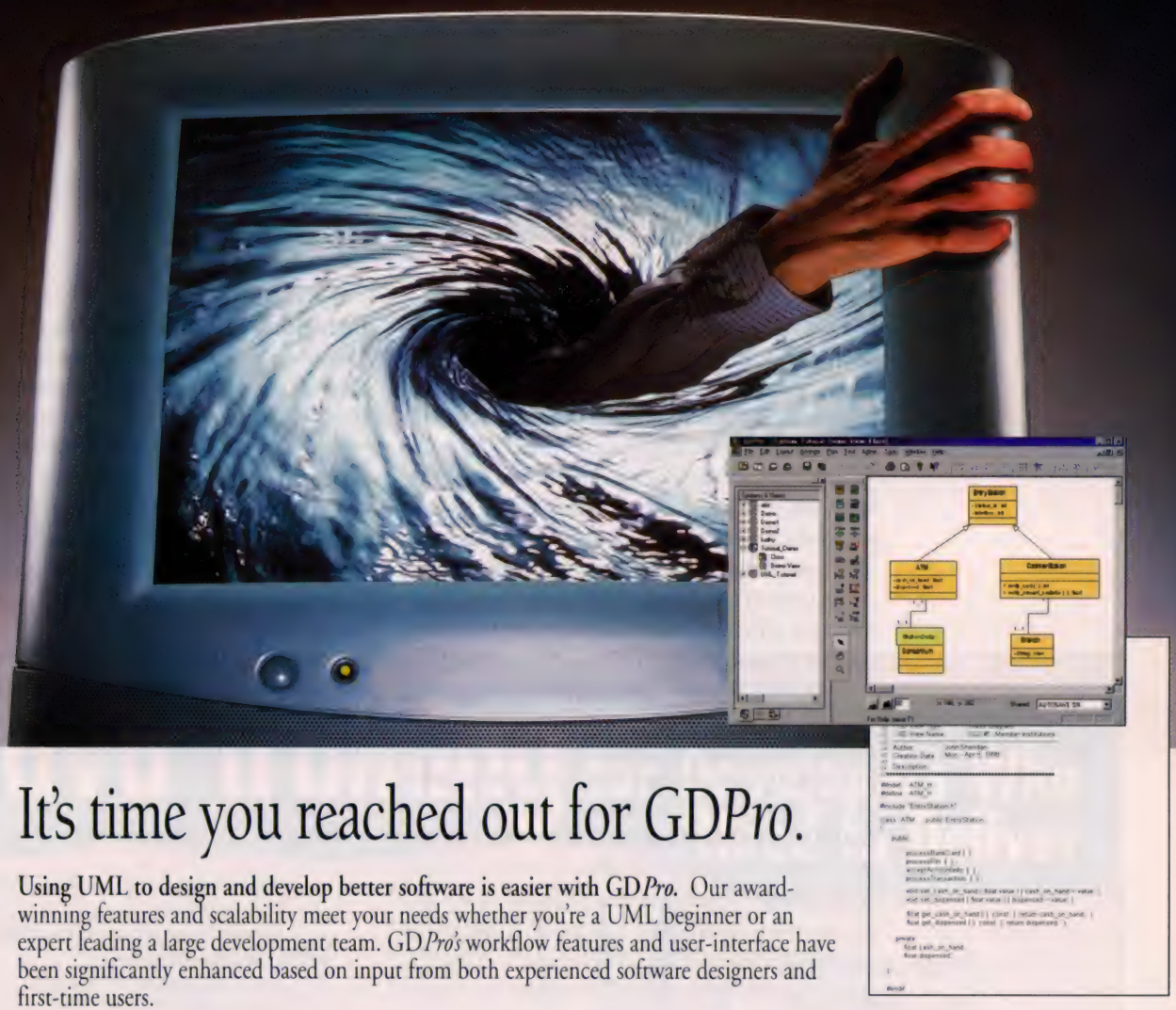
The CollationElementIterator class is intended for use in locale-sensitive text searching. However, it is missing several methods in JDK 1.1 that make it impossible to use with fast string searching algorithms such as Boyer–Moore. The following new methods were added in Java 2 to fix this:

1. The getOffset() method tells where a collation element was found.
2. The previous() and setOffset() methods enable backing up and moving around in the text being searched.
3. The new setText() method allows reuse of a CollationElementIterator. When collating or searching a large number of strings, it is much faster to reuse one CollationElementIterator than to construct a new one each time.
4. The isIgnorable() method tells whether a collation element is ignorable.
5. The getMaxExpansion() method returns the maximum length of any expansion sequence producing a given character. A fast search algorithm needs to know the maximum "shift" distance in looking for possible match sites. This is complicated by the fact that in natural language, a match can occur with different numbers of characters. If a search pattern for German text contains "oe", for example, it can match the single character "ö" in the text being searched. With the maximum expansion length, a fast search algorithm can compute the correct lower limit on shift distances.

Unicode Normalization

Unicode is more than just "wide ASCII." One of the principal operations on Unicode is to nor-

Are you asked to make your application faster, better, Internet-enabled, reusable, distributed, customer-oriented, component-based, "reengineered", *and* well-documented?



It's time you reached out for GD*Pro*.

Using UML to design and develop better software is easier with GD*Pro*. Our award-winning features and scalability meet your needs whether you're a UML beginner or an expert leading a large development team. GD*Pro's* workflow features and user-interface have been significantly enhanced based on input from both experienced software designers and first-time users.

The GD*Pro* tool kit includes all the features you need, including some that the competition simply can't match. Robust reverse-engineering and round-trip engineering are achieved without the use of cumbersome code markers. Enterprise developers love the collaborative "whiteboard" design facility and true team-based development support.

With GD*Pro*, you get a visual tool kit which works the way you do, not one which forces you to work around legacy tool technology. Plus you are not forced into a proprietary tool suite - GD*Pro* integrates with most other popular tools which you already use (like requirements management), or which you plan to use in your development environment.

When it comes to Software Analysis, Design, and Reengineering, you've got a lot at stake, and not all tools are what they appear to be. So you owe it to yourself...

Reach out for GD*Pro* 3.0 for 30 days...at no charge!

Call 800-811-ASTI (2784) today, or visit our web site at: www.advancedsw.com, or email us at: info@advancedsw.com

ADVANCED
SOFTWARE TECHNOLOGIES INC.

"Best Tools. Best Service. No Compromise."

malize text, ensuring that you have a unique spelling for a given text. Text normalization includes decomposition and composition forms of characters. Text can be normalized to be a *canonical* equivalent to the original unnormalized text or to be a *compatibility* equivalent to the original unnormalized text. For more information, please see Unicode technical report #15 on the Unicode Web site.

One of the Unicode normalization forms is used internally as a part of JDK 1.1, but it is not public. The Normalizer class incorporates this technology and allows either batch or incremental normalization of text. This class is provided on the alphaWorks Web site.

FORMATTING AND PARSING

JDK 1.1 provides a rich set of functionality for formatting values into strings and parsing strings into values in a locale-sensitive way. These include numbers, dates, times, and messages.

Number formatting supports spreadsheet-style patterns. For example, a format such as "#,##0.00#" will produce output like "1,234.567" or "5.00"; the pattern specifies that you have at least 2 decimal digits, but no more than 3. You can also reset the decimals and other characteristics of the pattern programmatically. Number formatting also provides powerful pattern parsing support for proportional font decimal alignment.

Date/Time formatting supports similar features, and is fully integrated with Calendar. Message formatting allows access to number, date, and time formatting within the context of a localizable string.

Substitutable Currencies

NumberFormat provides the factory method getCurrencyInstance(), which creates an object that can convert numbers to and from strings in the currency format of a given locale. In JDK 1.1, these formats were treated just like any other number formats. They were constructed from strings that were fetched from ResourceBundles. In 1.2, the currency symbol can be specified independently from the rules for decimal places, thousands separator, and so on, and is supplied in the pattern with the international currency symbol ("¤" = "\u00A4"").

```
// In Java 2
DecimalFormatSymbols us_syms = (DecimalFormat
  )fmt.getDecimalFormatSymbols();
us_syms.setCurrencySymbol("US$ ");
fmt.setDecimalFormatSymbols(us_syms);
result = fmt.format(1234.56)
// result is "US$ 1,234.56"
```

ISO Currency Codes

Additionally, we added an API to retrieve the 3-letter international currency codes defined in ISO 4217. These are necessary in an application that deals with many different currencies because the regular, one-character currency symbols are often shared by many different currencies. For example, both the US and Canada use "$" in their default currency format. An application dealing with both currencies will probably want to use "USD" and "CAD" instead. In Java 2, this is now possible, using a sequence of two international currency symbols ("¤¤" = "\u00A4\u00A4") in the pattern.

```
// In Java 2:
fmt = new DecimalFormat("\u00a4\u00a4 #,##0.00;(
    \u00a4\u00a4 #,##0.00)");
result = fmt.format(1234.56);
// result is "USD 1,234.56".
```

Parse Error Information

The abstract method parseObject() in java.text.Format is used to parse strings and turn them into objects. In JDK 1.1, the program can find out how far the parse got so that it can continue from that point on. However, it cannot find out how far it got if there was an error. In Java 2 a new field, errorOffset, now contains that information. If an error occurs during parsing, the formatters set this value before returning an error or throwing an exception.

In the following example, a text field is parsed for a number. If an error is found, the text beyond the error is highlighted, a message is displayed, and a beep is played.

```
// In Java 2:
String contents = textField.getText();
try {
        NumberFormat fmt = NumberFormat.
          getInstance();
        Number value = fmt.parse(contents);
} catch (ParseException foo) {
        errorLabel.getToolkit().beep();
        errorLabel.setText(
        myResourceBundle.getString("invalid number"));
        textField.select(foo.getErrorOffset(),
        contents.length());
}
```

Number Format Enhancements

On the alphaWorks Web site we provide a class that correctly supports exponentials in number formatting and parsing. The new number formatter supports formats such as "1.2345E3", as well as engineering exponents, where the exponent is always a power of 3. It also supports formatting and parsing BigInteger or BigDecimal values without loss of precision, and "nickel-rounding": the ability to round to multiples of a specified number, such as $0.05. (This is important for some countries whose smallest coin

is 5 units instead of 1. The implementation is not restricted to nickels, however, and can be used to round to multiples of any given value.)

Here is an example using the class from the alphaWorks Web site.

```
NumberFormat fmt = new NumberFormat("0.0000E00");
String result = fmt->format(123456789);
// result is "1.2346E08"
```

Number Formats in Words

The ability to take a numeric value (e.g., 12,345) and translate it into words (e.g., "twelve thousand three hundred forty-five") is often needed in business applications, for example, to write out the amount on a check. Number spellout in English is a relatively easy thing to do; good algorithms for this are well-known and widely used. A number-spellout engine that can be customized for any language is another thing altogether.

It's not enough to simply take the algorithm for English and read the literal string values from a resource file. English separates all component parts of a number with spaces; Italian and German do not. Some languages, such as Spanish and Italian, drop the word for "one" from the phrases "one hundred" or "one thousand." There are many other examples that show translating a number into words is not a trivial task.

To solve these issues, we developed a class called RuleBasedNumberFormat. It's a general, rule-based mechanism for converting numbers to spelled-out strings. This class is available on the alphaWorks Web site, along with information on the usage and rule syntax.

```
RuleBasedNumberFormat fmt =
    new RuleBasedNumberFormat(rules);

String result = fmt.format(1234);
// result is "one thousand two hundred thirty four"
```

CONCLUSION

The internationalization services in Java 1.1 provide a wide range of functionality and are easily extended to add additional features and to support additional countries. We have had an opportunity to discuss some of the design decisions taken in developing these classes and some of the enhancements that are being included in future releases. A more detailed discussion is available at the IBM international classes Web site, and includes more about the JDK i18n classes and possible future interna-

Java Training for Developers Who Need to Deliver

Petronio Technology Group's industry-acclaimed Java training courses can help accelerate your Java learning curve and keep you abreast of today's latest Java technologies. Advanced topics include JDBC, Beans, I18N, Security, JNI, RMI and Serialization. Popular hands-on Java training workshops include:

- Ultimate Java Programming
- Advanced Programming
- JavaBeans
- JFC

* Public Training Coming to a Location Near You. See www.petronio.com/locator for details.

Now offering JavaBeans Java 1.2 & JFC

FREE Java Quick Reference Card

Receive a *free* copy of our exclusive Java Quick Reference Card containing valuable programming information including an overview of Java syntax, Java keywords, an operator precedence table, exception handling syntax and more. Visit our web site at www.petronio.com/refjr to request your free copy.

"Outstanding instruction and customized course materials. The mix of lecture and labs gave us a great jump start in meeting our C++ and Java needs."
Mark Romagna, Regional Technical Manager, Sybase, Inc. - Powersoft Business Group

"I highly recommend Petronio Technology Group's Java courses to any professional development team that needs to quickly come up to speed on Java."
Keson Khieu, R&D Project Manager, Hewlett Packard

PETRONIO Technology Group
Targeted Technology Training

394 Lowell Street, Suite Two ■ Lexington, MA 02173 ■ 781-778-2000 ■ FAX 781-778-4000 ■ www.petronio.com ■ info@petronio.com

tionalization improvements that IBM is discussing with Sun. These future possibilities include the following:

- A character conversion API, so that you can actually find out which character code converters are supported on your system, and get understandable display names for them, e.g., instead of "Cp1089", in English displaying Arabic (ISO 8859-6) and in French displaying Arabe (OSI 8859-6)
- Customized locales, so that you could have fine-grained control over default behavior, e.g.,

```
NumberFormat.setInstance(new Locale("en", "US",
"Acme Widgets, Inc."),
                    new DecimalFormat("#,##0.0#"));
```

- Convenience methods for common cases, e.g.,

```
String value = Number.formatCurrency(amount);
```

We are working on many future enhancements; some of which are available right now on IBM's alphaWorks Web site. We encourage those interested to download versions from there—any comments on the design and implementation are welcome!

ACKNOWLEDGMENTS

Our thanks to Kathleen Wilson, Rich Gillam, and Laura Werner for their extensive review, suggestions, and for organization of the document. Many other people at IBM and Sun contributed to the Java internationalization efforts. ■

Dr. Mark Davis is a program director at IBM's Java Technology Center Silicon Valley, CA, co-founder of the Unicode effort, and is president of the Unicode Consortium. He can be contacted at mark@macciato.com. Helena Shih is the technical lead of the IBM Classes for Unicode at IBM's Center of Java Technology. She can be contacted at hshih@us.ibm.com.

Trademarks

Unicode is a trademark of Unicode, Inc. Copyright © 1998, IBM Corp. All rights reserved.

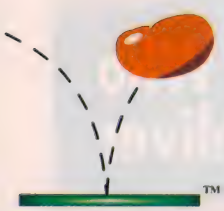
URLs

alphaWorks
http://www.alphaWorks.ibm.com/

Unicode internationalization services
http://www.ibm.com/java/tools/international-classes/

ISO codes
http://www.unicode.org/unicode/onlinedat/

Unicode technical report #15
http://www.unicode.org/unicode/reports/tr15/



Mobilize the Enterprise™ *with* Jumping Beans™

Fully exploit the capabilities of Java with Jumping Beans, an intuitive and powerful mobility framework allowing never-before possible solutions. With the Jumping Beans framework, a developer can quickly create applications which dynamically move from one computer to another during execution, in a secure, robust fashion.

Jumping Beans enables many new solutions in:

- Network management
- Device management
- Workflow
- Database access
- Business process implementation
- Data gathering
- much more

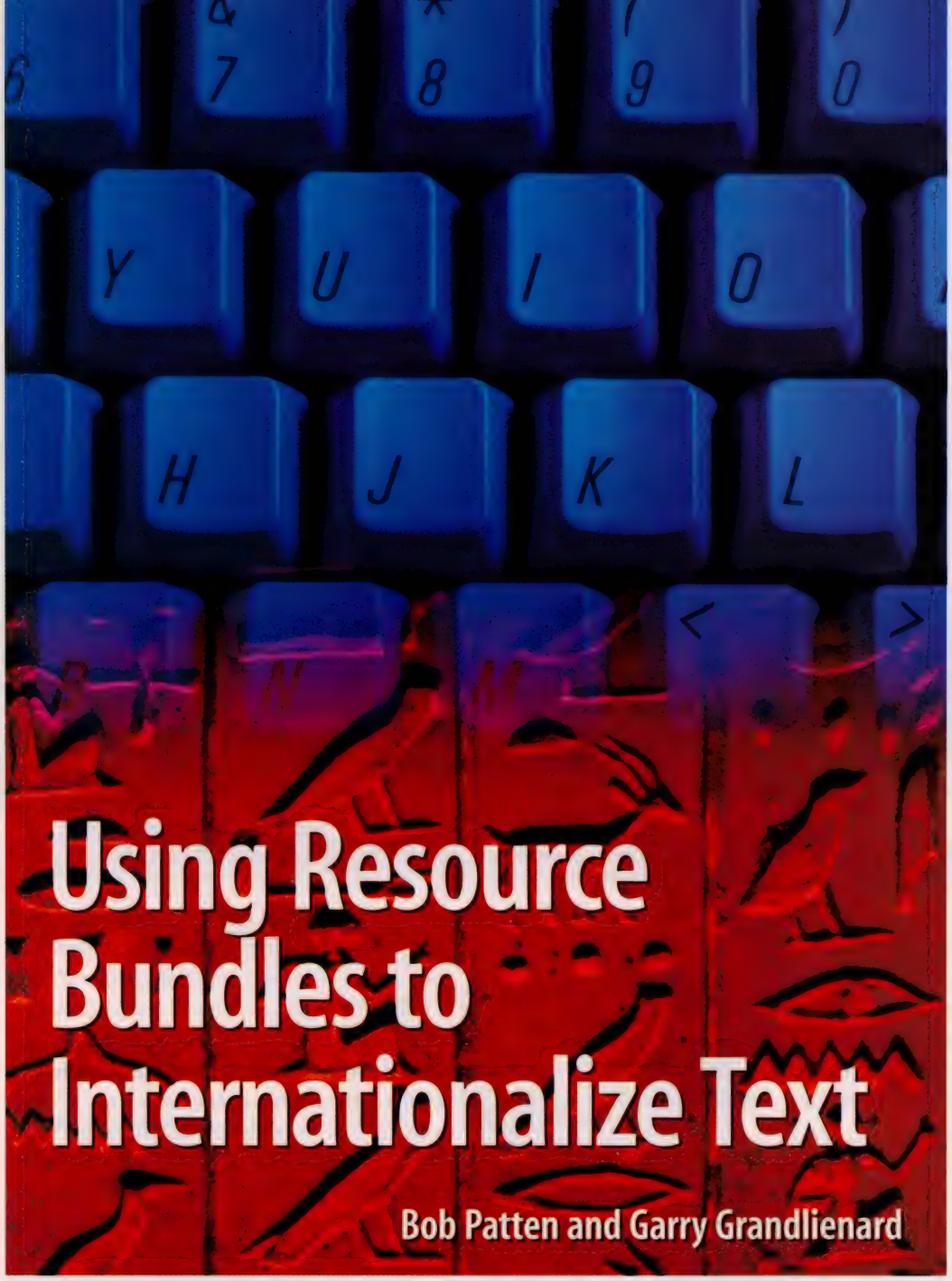
Full-featured evaluation copy at:
www.JumpingBeans.com
or call 408-738-4616

Selected Best of Show
1998 SIGS Conference for Java Development

Jumping Beans is ideal for enterprise computing, with industrial-grade features including:

- Centralized management
- Solid security designed-in
- Mobile CORBA
- Mobilizes the entire application, including code, state, data, resources, images, JARs, etc.
- Store-and-forward with guaranteed delivery
- 126 Kb footprint, PersonalJava compliant
- others

"Jumping Beans", "Mobilize the Enterprise", and the Bouncing Bean Logo are trademarks of Ad Astra Engineering, Inc.
Java, 100% Pure Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.



Rob Atkins/Image Bank

Java and multi-language applications

Using Resource Bundles to Internationalize Text

Bob Patten and Garry Grandlienard

INTERNET APPLICATIONS ARE, by definition, graphical in nature. As a result, you end up with a large number of strings due to text on buttons, messages, dialogs, labels, lists, and so forth. Most applications today need to support the ability to quickly switch between languages, for this text to support the Internet's worldwide community. This results in having to support multiple versions of an application and adds a burden on the developer unless forethought is given on how to handle this large number of strings. Luckily, Java supplies several ways in which these strings can be handled.

However, internationalizing an application in Java is no simple matter. While the language gives you a variety of ways of handling these strings, the burden is placed on the developer to find the most effective way to use the facilities that Java provides. Recently, we had the opportunity to study this problem while implementing a multi-language application for the banking industry. We will briefly discuss the two most common ways of supporting multiple string values, point out the problems with each, and then discuss a way that allows the developer to quickly handle strings with no extra work.

Lets take a normal program that creates a set of buttons and specifies their labels. For instance, many Java programs add buttons to a panel using a technique such as,

```
add( new Button("OK") );
add( new Button("Cancel") );
```

Then in the listener methods, the program has to verify which button has been pressed by doing something similar to:

```
if ( button.equals("OK") )
{
        // do something
}
else if ( button.equals("Cancel") )
{
        // do something else
}
```

Note that the determination of which button was pressed is based on the label. If multiple languages are supported this leads to a huge maintenance headache because there are duplicate strings in several places and, therefore, a large amount of work is required to internationalize the application. The first thing that we can do to help as developers is to keep track of the button reference and use that to determine which button was pressed, instead of using the button label. For instance, the above code could be changed to:

```
okButton = new Button("OK");
cancelButton = new Button("Cancel");
```

Then in the listeners we would code something like:

```
if ( button == okButton)
{
        // do something
}
else if (button == cancelButton)
{
        // do something else
}
```

This strategy allows us to only deal with the actual string value in a single location, namely, when we create it. Note: some IDEs will generate the former code and some will generate the latter. Examples within this article use IBM's VisualAge for Java (which generates the latter, and in my mind, the correct code). We could then use a list resource bundle to localize all the strings into a single section of code. For example, we may code something similar to:

```
public class ScreenStrings extends ListResourceBundle
{
        public Object[][] getContents() { return
    contents; }
        static final Object[][] contents =
            {   { "okButton", "OK" },
                { "cancelButton", "Cancel" } }
}
```

Here we extend the ListResourceBundle, which provides the basis for handling our resource strings. The getString method is then used to obtain the string value based on the key. The code for the creation of the buttons would be changed to:

```
okButton = new Button(
primaryScreenStringsObject.getString("okButton") );
cancelButton = new Button(
primaryScreenStringsObject.getString("cancelButton") );
```

Notice that instead of specifying the labels directly we use the button's key. This allows us to localize the strings into a single class. However, this does have a drawback. Usually a non-technical person will be the one that performs the translation. Having this person modify our source can lead to unintentional syntax errors. A better solution would be to place these strings into a resource bundle file and use the PropertyResourceBundle class to access this file. Using this scheme we would wind up with:

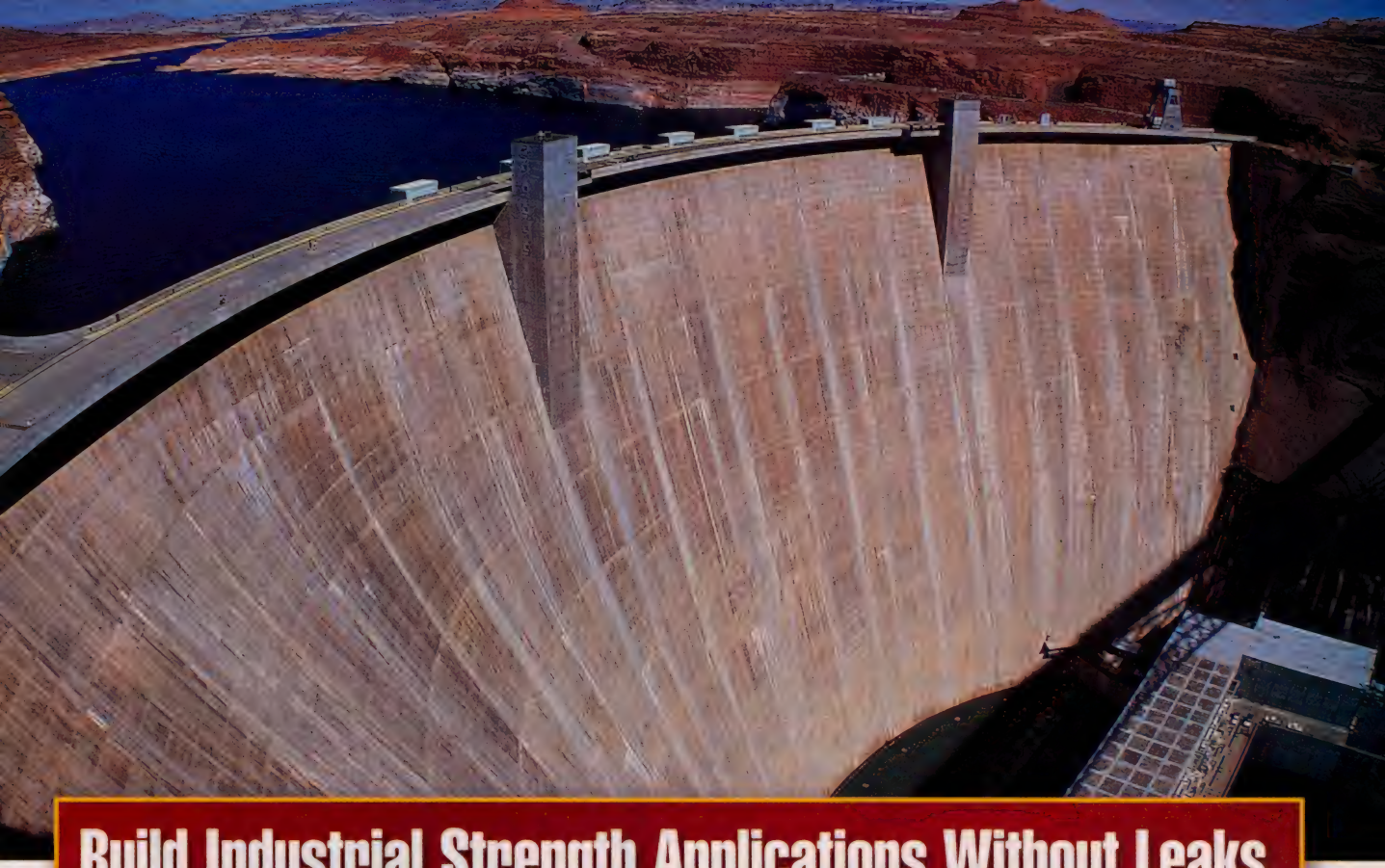
```
public class ScreenStrings extends
   PropertyResourceBundle
{
        ScreenStrings() throws IOException
        {
        super(ScreenStrings.class.getResourceAsStream
                ("PrimaryScreenStrings.resource") );
        }
}
```

Then creating the PrimaryScreenStrings.resource file to contain:

```
okButton=OK
cancelButton=Cancel
```

We can provide this file to the non-technical translator (which has a straightforward syntax), and s/he can translate the actual strings (right side of the equal sign). As the number of resource bundle files grow in proportion to the number of screens or languages, we simply supply all the files to the translator.

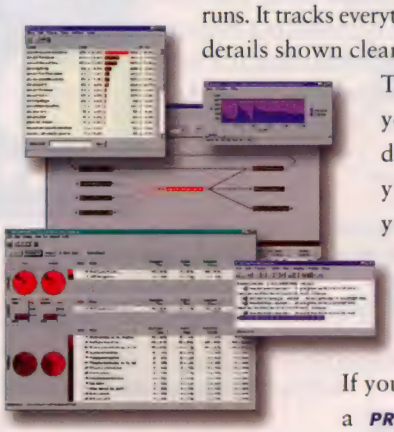
This allows us, as developers, to insulate our code from changes to the strings and from mistakes during the translation. However, as wonderful as this is, it is really only feasible in the most simplistic of cases. Why, you ask? Well, because if we use this scenario we have to have a separate class for each resource bundle that we want to access. So, if we have 10 screens with a resource bundle per screen we wind up with 10 classes. If we have two combo boxes (containing text phrases) for each screen we would wind up with two additional classes per screen, which would result in 20 additional classes or 30 total classes. While these classes are basically the same (only the name of the class and the name of the resource bundle file changes), this approach can lead to a lot of additional work.



Build Industrial Strength Applications Without Leaks

Whether you're building a high performance hydro-electric dam or a ***MISSION-CRITICAL JAVA APPLICATION***, you know that those pesky leaks can sabotage everything! You need a professional tool that is accurate, easy to use and helps you deliver ***SCALABLE, ROBUST PERFORMANCE*** from start to finish.

JPROBE PROFILER is used by Java developers around the world to improve the performance of their Java applications. And now, the most ***POWERFUL TOOL*** for finding and eliminating Java performance bottlenecks makes it easy for you to find and plug memory leaks in your Java code. Use the new ***HEAP REFERENCE GRAPH*** to instantly see a heap snapshot analysis and untangle even the most complex object references that prevent garbage collection.



JPROBE Profiler

JProbe Profiler gives you the most ***ACCURATE MEASUREMENT*** of both time and memory consumption while your program runs. It tracks everything for you, thread by thread, with method details shown clearly right back to each line of source code*. The advanced ***CALL GRAPH*** helps you analyze your profiling snapshot and quickly drill-down to performance bottlenecks. And you'll get superior performance where you want it because JProbe Profiler provides ***MULTI-PLATFORM*** support for Windows NT and Solaris in JDK1.1, JDK1.2, and JIT environments.

If you develop Java code for a living, you need a ***PROFESSIONAL JAVA PROFILER.*** Get a free evaluation copy of ***JPROBE PROFILER*** right now and start making your Java code industrial strength, without leaks.

Try JProbe Now! ***www.klgroup.com/jprobe***

The Easiest Way to Search and Destroy Java Performance Bottlenecks

KL GROUP

Software Development Productivity

KL Group Inc. Toll-Free: 1-888-361-6205 Tel: 416-594-1026 Fax: 416-594-1919

KL Group Europe B.V. Tel: +31 (0)20 679 95 03 Fax: +31 (0)20 470 03 26

www.klgroup.com

JAVA

* Source code not required to profile with JProbe. JProbe is a trademark of KL Group Inc. All other products are trademarks or registered trademarks of their respective companies. JR003

Using Resource Bundles to Internationalize Text

There has to be a better solution. This is where most texts and examples stop; so we are left with devising an alternate solution. It is not uncommon to have an application with more than 20 screens, each potentially having several choice boxes; the result is A LOT of resource classes all doing the same thing. When trying to come up with a solution, one might think that we can simply extend the PropertyResourceBundle and pass in the resource bundle file name as an argument to the constructor. However, the PropertyResourceBundle class requires us to either have a default constructor or use the constructor that passes in an InputStream. Using the default constructor defeats the purpose, because we cannot pass in the resource bundle name, and using the other constructor requires the calling code to perform work that should be performed within this class. Attempting to extend the PropertyResourceBundle and obtain the resource bundle via a variable works no better, (Note: the invocation of the parent's constructor, e.g.,: the super call above, must be the *first* line within the constructor). Because all of these techniques either do not work or defeat the purpose of having a single class we are left to find a different solution.

By making a fairly simple change in the implementation we can reduce all resources' classes, regardless of the number of resource bundles, into a single class. The solution is to create a class that *contains* a PropertyResourceBundle instead of *extending* it. Our code will then look like (note: try/catch blocks have been removed to aid in readability):

```
class AppResources
{
        PropertyResourceBundle bundle;
        public AppResources( String aResourceName )
        {
            aResourceName = aResourceName + ".resource";
InputStream in = AppResources.class.
 getResourceAsStream(aResourceName);
            bundle = new PropertyResourceBundle( in );
        }
        public getString( String aKey )
        {
            return bundle.getString( aKey );
        }
}
```

If this is extended to support multiple languages (e.g.: multiple locales), the above class could be changed to:

```
class AppResources
{
        PropertyResourceBundle bundle;
        public AppResources( String aResourceName )
        {
            aResourceName = aResourceName + "_" +
                        getCurrentLocale.toString() + ".resource";
InputStream in = AppResources.class.
 getResourceAsStream(aResourceName);
            bundle = new PropertyResourceBundle( in );
        }
        public getString( String aKey )
        {
            return bundle.getString( aKey );
        }
}
```

Then, by creating resource bundle files with like names:

```
PrimaryScreenStrings .resource
PrimaryScreenStrings_fr_CA .resource
PrimaryScreenStrings_fr_FR.resource
```

any language can be supported for a component's label without any change to the code.

Using this scheme, the creation of the buttons would result in code similar to:

```
AppResources primaryScreen = new AppResources(
    "PrimaryScreenStrings" );
okButton = new Button( primaryScreen.getString(
    "okButton" ) );
cancelButton = new Button( primaryScreen.getString(
    "cancelButton") );
```

This allows the "correct" language for the button's label to be automatically loaded just by setting the locale correctly. Furthermore, if we make use of a JDK 1.1 feature allowing components to be named, then we can change this to:

```
okButton = new Button();
okButton.setName("okButton");
cancelButton = new Button();
cancelButton.setName("cancelButton");
```

Then at anytime the label can be changed by:

```
okButton.setLabel( primaryScreen.getString(
  okButton.getName() ) );
cancelButton.setLabel( primaryScreen.getString(
  cancelButton.getName() ) );
```

and we do not even have to keep track of the component button's key. (Note: this is a time saving use of the *name* property especially if your IDE automatically sets this property as VisualAge for Java does). The *name* property was introduced in 1.1 to remove the dependency from relying on the label of a component. Most IDEs will support this in the future it yours does not currently. To change to another language, we simply issue the code:

```
setCurrentLocale( new Locale( "fr", "CA" );
primaryScreen = new AppResources(
  "PrimaryScreenStrings" );
okButton.setLabel( primaryScreen.getString(
  okButton.getName() ) );
cancelButton.setLabel( primaryScreen.getString(
  cancelButton.getName() ) );
```

The World's Largest E-Business and IP Technology Event

ESSENTIAL ACCESS

TO THE VERY BEST INTERNET TOOLS, TALENT AND TECHNOLOGY

THE SIXTH ANNUAL

SPRING Internet World 99

Conference: April 12-16
Exhibits: April 14-16
Los Angeles Convention Center

- **100** in-depth conference sessions
- **18** vertical mini-conferences
- **250** A-list speakers
- **500** exhibiting companies
- **12+** acres of Internet systems, products, and services
- **5** dynamic keynotes
- **40,000** Internet professionals

PRODUCED BY:

Penton

SPONSORED BY:

internet WORLD

internet.com

BOARDWATCH MAGAZINE

SHOW NETWORK PROVIDED BY:

IBM

From the E-Mail Center to the Show floor, IBM puts the "Net" into Internet World.

FOR CONFERENCE AND EXHIBITION INFORMATION:
Visit: http: www.internet.com
Call: 1-800-500-1959
e-mail: siwprogram@mecklermedia.com

TO EXHIBIT:
Call Sean Moriarty at 203-341-2969 or
e-mail: smoriarty@mecklermedia.com

Code: N61

SAVE $50. Bring this ad to the show for FREE exhibit hall admission. For the trade only.
Business card required for admission. No one under 18 admitted.

Listing 1.

How to implement an application to dynamically switch languages.

```
Sample code:
// To Do List Application
package ToDoList;
import java.awt.*;
import java.util.*;

public class ToDoList extends Panel implements
  java.awt.event.ActionListener, java.awt.event.ItemListener {
    private Button ivjAddItemButton = null;
    private CheckboxGroup ivjCheckboxGroup1 = null;
    private Checkbox ivjEnglishCB = null;
    private Checkbox ivjFrenchCB = null;
    private List ivjList1 = null;
    private Button ivjRemoveItemButton = null;
    private TextField ivjTextField1 = null;
    private Label ivjToDoItemLabel = null;
    private Label ivjToDoListLabel = null;

public ToDoList() {
    super();
    initialize();
}

public ToDoList(java.awt.LayoutManager layout) {
    super(layout);
}
public void actionPerformed(java.awt.event.ActionEvent e) {
    if ((e.getSource() == getAddItemButton()) ) {
        conn0(e);
    }
    if ((e.getSource() == getRemoveItemButton()) ) {
        conn2(e);
    }
}

public void changeChildren(Panel panel, AppResources appResources)
{
    Component children[] =panel.getComponents();
    for (int i=0; i<children.length; i++)
    {
        if (children[i] == null)
        {
            continue;
        }

        if (children[i] instanceof Label)
        {
            Label label = (Label)children[i];
            label.setText(appResources.getString(label.getName()) );
        }
        else if (children[i] instanceof Button)
        {
            Button button = (Button)children[i];
            button.setLabel(appResources.getString(button.getName())
);
        }
        else if (children[i] instanceof Panel)
        {
            changeChildren( (Panel)children[i],appResources);
        }
    }

    return;
}
private void conn0(java.awt.event.ActionEvent arg1) {
    try {
        getToDoList().addItem(getToDoItem().getText());
    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
    }
}

private void conn2(java.awt.event.ActionEvent arg1) {
    try {
        getToDoList().remove(getToDoList().getSelectedItem());
    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
    }
}

private void conn4SetTarget() {
    try {
        getEnglishSelection().setCheckboxGroup(
            getLanguageSelectionGroup());
    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
    }
}

private void conn5SetTarget() {
    try {
        getFrenchSelection().setCheckboxGroup(
            getLanguageSelectionGroup());
    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
    }
}

private void conn6(java.awt.event.ItemEvent arg1) {
    try {
        this.switchToFrench();
    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
    }
}

private void conn7(java.awt.event.ItemEvent arg1) {
    try {
        this.switchToEnglish();
    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
    }
}

private Button getAddItemButton() {
    if (ivjAddItemButton == null) {
        try {
            ivjAddItemButton = new java.awt.Button();
            ivjAddItemButton.setName("AddItemButton");
            ivjAddItemButton.setBounds(60, 345, 125, 30);
            ivjAddItemButton.setLabel("Add Item");
        } catch (java.lang.Throwable ivjExc) {
            handleException(ivjExc);
        }
    };
    return ivjAddItemButton;
}

private CheckboxGroup getLanguageSelectionGroup() {
    if (ivjCheckboxGroup1 == null) {
        try {
            ivjCheckboxGroup1 = new java.awt.CheckboxGroup();
        } catch (java.lang.Throwable ivjExc) {
            handleException(ivjExc);
        }
    };
    return ivjCheckboxGroup1;
}

private Checkbox getEnglishSelection() {
    if (ivjEnglishCB == null) {
        try {
            ivjEnglishCB = new java.awt.Checkbox();
```

Now Even More AMAZING!

Are You Ready for the Next BIG THING in Java Development???

jtest! is the only tool on earth which finds your uncaught runtime exceptions for you!

But don't take our word for it. Try it for yourself!!!

Watch as **jtest!** goes through your code, automatically finding input to flush your bugs!

Marvel as it reports each exception found!

Wonder how you ever got along without it!!

jtest! works on:
Applets
Applications
API's
JavaBeans
any Java Code!

Find Hidden Bugs in Java Classes!

jtest

FREE Download

www.parasoft.com/jtest

1-888-305-0041

ParaSoft Corporation

Listing 1. (continued)

```
            ivjEnglishCB.setName("EnglishCB");
            ivjEnglishCB.setBounds(61, 393, 125, 30);
            ivjEnglishCB.setLabel("English");
        } catch (java.lang.Throwable ivjExc) {
            handleException(ivjExc);
        }
    };
    return ivjEnglishCB;
}

private Checkbox getFrenchSelection() {
    if (ivjFrenchCB == null) {
        try {
            ivjFrenchCB = new java.awt.Checkbox();
            ivjFrenchCB.setName("FrenchCB");
            ivjFrenchCB.setBounds(247, 393, 125, 30);
            ivjFrenchCB.setLabel("French");
        } catch (java.lang.Throwable ivjExc) {
            handleException(ivjExc);
        }
    };
    return ivjFrenchCB;
}

private List getToDoList() {
    if (ivjList1 == null) {
        try {
            ivjList1 = new java.awt.List();
            ivjList1.setName("List1");
            ivjList1.setBounds(17, 87, 390, 223);
        } catch (java.lang.Throwable ivjExc) {
            handleException(ivjExc);
        }
    };
    return ivjList1;
}

private Button getRemoveItemButton() {
    if (ivjRemoveItemButton == null) {
        try {
            ivjRemoveItemButton = new java.awt.Button();
            ivjRemoveItemButton.setName("RemoveItemButton");
            ivjRemoveItemButton.setBounds(245, 345, 125, 30);
            ivjRemoveItemButton.setLabel("Remove Item");
        } catch (java.lang.Throwable ivjExc) {
            handleException(ivjExc);
        }
    };
    return ivjRemoveItemButton;
}

private TextField getToDoItem() {
    if (ivjTextField1 == null) {
        try {
            ivjTextField1 = new java.awt.TextField();
            ivjTextField1.setName("TextField1");
            ivjTextField1.setBounds(104, 14, 291, 31);
        } catch (java.lang.Throwable ivjExc) {
            handleException(ivjExc);
        }
    };
    return ivjTextField1;
```

Of course, the aformentioned code would force the previous language resources to have to be completely reloaded but caching the previous languages as needed (to speed up the switching of languages) can solve that.

Listing 1 shows how to implement an application to dynamically switch languages. (Note: the example was developed using IBM's VisualAge for Java product and has been tested under JDK 1.1.5.)

Labels for GUI components are not the only things that normally have to change when the language changes. There are also the items within choice boxes (given that they are strings and not numerical) and the prompts (also known as long, fly over help). This same basic scheme can be used to handle these cases as well. For instance, if we need to also update the choices within a choice box then, when we create the choice box we make sure to set the "name" of the choice box (if your IDE does not do it for you). While we could store the items for a choice box within the same resource bundle as the labels it is easier to place them in a separate resource bundle file. Why? By storing them in a separate resource bundle file, we can load the entire list of items by using an enumeration. For instance:

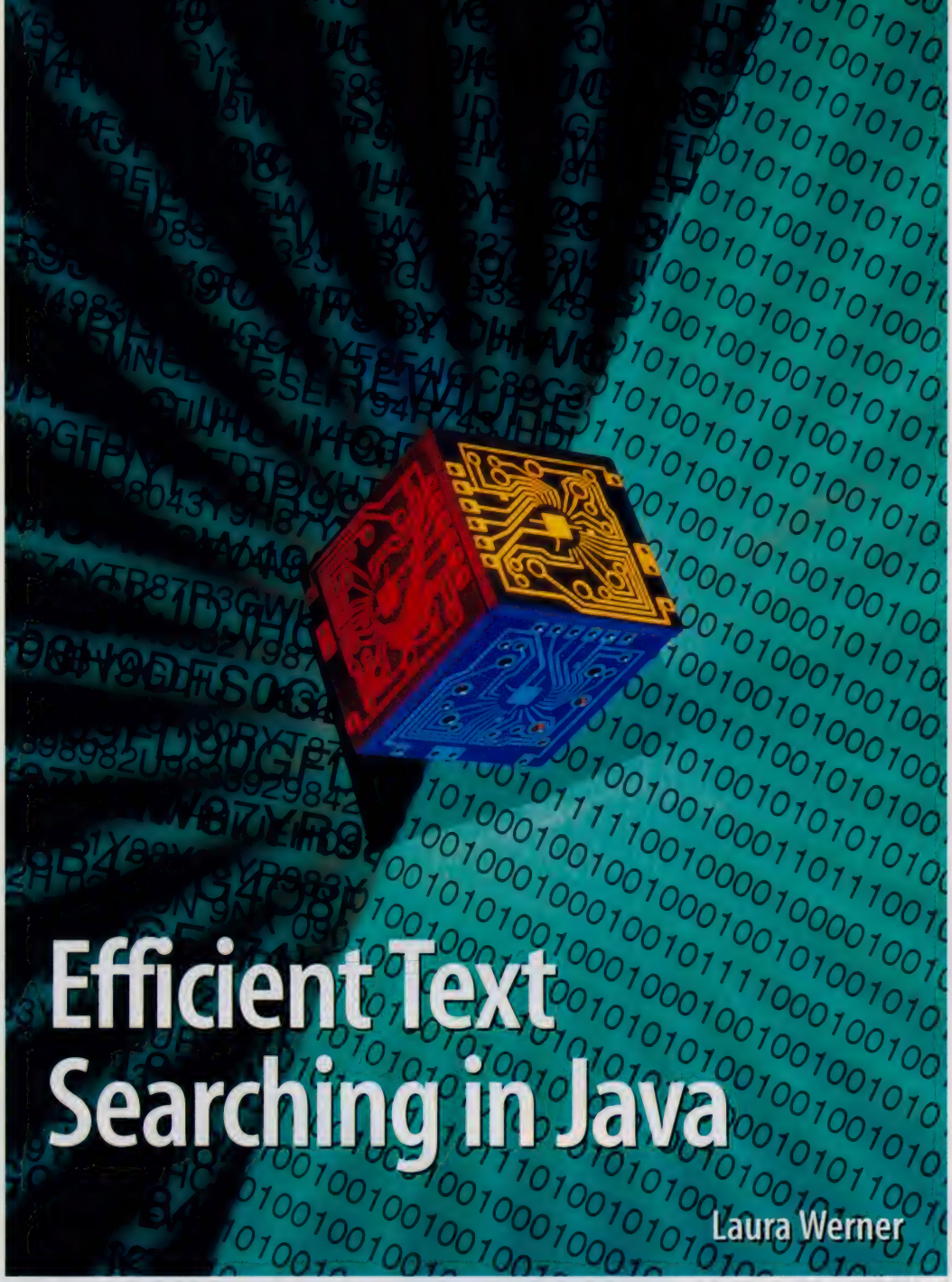
```
countryCBItems = new AppResources(
  countryChoiceBox.getName() );

Enumeration en = countryCBItems.getKeys();
while (en.hasMoreElements())
{
        String key = (String)en.nextElement();
        String value = countryCBItems.getString(key);
        addItem(value);
}
```

Notice that in this code we used the name of the component as the name for the resource bundle that allows a single name to be used for all aspects of the component; from label, to items within a choice box, to fly over help.

The proper use of resource bundles can expedite the release of an application supporting multiple languages and seriously reduce the amount of coding required from us. While we've only focused on the issue of internationalizing strings, there is much more that has to be done to create a true international application; displaying text right to left or left to right and kanji symbols to name just a few. However, this is a first step and by making use of the extremely flexible and powerful resource bundle facility within Java, we can provide multiple language capability for our applications with little extra effort. ■

Bob Patten is a Senior I/T Specialist with the IBM Enterprise Java Services National Practice in RTP, NC. He can be contacted at wrpatten@us.ibm.com.

Garry Grandlienard is a Senior Consultant with CrossLogic Corporation in Raleigh, NC. He can be contacted at ggrand@crosslogic.com.



Pierre-Yves Goavec/Image Bank

Efficient Text Searching in Java

Laura Werner

Sorting and searching international text may seem hopeless, but the JDK 1.1 class java.text.Collator is here to help

TEXT SEARCHING AND sorting is one of the most well researched areas in computer science. It is covered in an introductory algorithms course in nearly every engineering school, and there are entire books devoted to the subject. Why then, you might ask, is it necessary to publish yet another article about searching?

The answer is that most of the well-known, efficient search algorithms don't work very well in Unicode, which includes the char type in Java. Algorithms such as Knuth–Morris–Pratt and Boyer–Moore utilize tables that tell them what to do when a particular character is seen in the text being searched. That's fine for a traditional character set such as ASCII or ISO Latin-1 where there are only 128 or 256 possible characters.

Java, however, uses Unicode as its character set. In Unicode, there are 65,535 distinct characters that cover all modern languages of the world, including ideographic languages such as Chinese. In general, this is good; it makes the task of developing global applications a great deal easier. However, algorithms like Boyer–Moore that rely on an array indexed by character codes are very wasteful of memory and take a long time to initialize in this environment.

And it gets worse. Sorting and searching non-English text presents a number of chal-

lenges that many English speakers are not even aware of. The primary source of difficulty is accents, which have very different meanings in different languages, and sometimes even within the same language:

- Many accented letters, such as "é" in "café", are treated as minor variants on the letter that is accented, in this case "e".
- Sometimes the accented form of a letter is treated as a distinct letter for the purposes of comparison. For example, "Å" in Danish is treated as a separate letter that sorts just after "Z".
- In some cases, an accented letter is treated as if it were two letters. In traditional German, for example, "ä" is compared as if it were "ae".

Other difficulties arise when multiple characters compare as if they were one, such as in traditional Spanish where "ch" is treated as a single letter that sorts just after "c" and before "d", or when single characters such as the "Æ" ligature or the German "ß" are treated as if they were spelled out as "AE" or "ss".

All of the above might make the problem of sorting and searching international text seem hopeless. While not impossible, it *is* a difficult problem. Do not despair, however. Starting in JDK 1.1, the class java.text.Collator is here to help.

COLLATORS MADE SIMPLE

For JDK 1.1, Taligent, which has since been absorbed into IBM, contributed a number of international support classes to the Java Class Libraries. These classes are now maintained and enhanced by engineers at the IBM Center for Java Technology in Cupertino, CA. In combination with work done by Sun, they provide a powerful foundation for developing truly global applications.

Among the new classes introduced in JDK 1.1 were Collator and RuleBasedCollator, both in the java.text package. Collator provides an abstract interface for comparing text, while RuleBasedCollator is a concrete subclass that implements the comparison using a rule-driven algorithm.

Using these classes to compare two strings is quite straightforward:

```
Collator c = Collator.getInstance();
if (c.compare(string1, string2) == 0) {
    // Strings are the same
}
```

TOPLink Does More

Choose the Object-Relational Mapping Tool used by Industry Leaders Around the World

TOPLink™ bridges the world of objects and the world of relational databases.

TOPLink allows you to map complex business objects to relational databases so you can leverage your investment in data and deploy new applications with Java.

TOPLink incorporates a meta-data architecture that separates the details of persistence from the business logic. *No SQL programming is required.*

Certified 100% Pure Java, **TOPLink** is used by Fortune 500 Java developers world-wide. It supports Java 2 and Enterprise JavaBeans (EJBs).

TOPLink features include:

- Flexibility with 10 different types of mappings
- 3-tier application server support
- Object-level transactions
- Queries in terms of objects, not SQL
- Object caching and identity

TOPLink's EJB support

includes:

- Transparent mapping of EJBs to relational tables
- Bean and/or container-managed entity bean persistence
- Fully integrated functionality with session beans
- Availability for a number of EJB servers, including WebLogic (formerly Tengah), Oracle, and others

For a **FREE 30-day evaluation** or more information, visit our web site or call:

(613) 225-8812

TOPLink™
THE OBJECT PEOPLE

TOPLink is a registered trademark of The Object People, Inc. Java and Java-based logos are trademarks or registered trademarks of Sun Microsystems, Inc. Other products referred to are trademarks of their respective owners

www.objectpeople.com

CANADA • U.S.A. • U.K. • GERMANY

THE OBJECT PEOPLE

Table 1. Example of the components of a collation element.

CHARACTER	PRIMARY	SECONDARY	TERTIARY
a	1	0	0
á	1	1	0
A	1	0	1
...			
B	2	0	1
...			
é	5	1	0

In a real program, of course, you would only call Collator.getInstance once and would then use the same collator object to perform as many comparisons as you wanted. Behind the scenes, getInstance determines the current default Locale, loads the appropriate collation rules, and constructs a Collator object that it returns. Currently, this object is always a RuleBasedCollator. However, in the future, different locales may require different classes with customized behavior, which is why you use a factory method to create instances of the more abstract Collator class.

UNDER THE HOOD

Internally, RuleBasedCollator.compare has an awful lot of bookkeeping to do. A byte-by-byte string comparison function like C's strcmp can walk though strings one character at a time and compare the bytes, but if Collator did something that simple it would rapidly get out of sync the first time it saw a contracting character like the Spanish "ch" or an expanding character like "Æ".

To keep track of this, RuleBasedCollator first translates strings into a series of *collation elements*, which correspond to single entities in the input string. In English, each character in the input maps to a collation element, but "Æ" produces two elements and the Spanish "ch" produces just one. This translation is done by the utility class CollationElementIterator, which uses a set of mapping tables built from the rules passed to the collator's constructor.

CollationElementIterator is a public class, and you can use it yourself to do searches, as we will see next. As an introduction, let's use it to iterate over the elements for a simple string:

```
RuleBasedCollator c =
(RuleBasedCollator)Collator.getInstance();
CollationElementIterator iter =
c.getCollationElementIterator("Foo");

int element;
while ((element = iter.next()) !=
CollationElementIterator.NULLORDER) {
        System.out.println("Collation element is: " +
                                Integer.toString(e,16) );
}
```

As you can see from this example, a collation element is a fairly simple creature; it's an int that describes where a character or group of characters falls in the sorting sequence. Higher-numbered elements are sorted after lower-numbered ones.

Of course, it's really a bit more complicated than that. Each collation element can be broken down into three components (also known as weights or orders): *primary*, *secondary*, and *tertiary*. The primary component of a collation element corresponds to which base letter the order represents, so "A" and "B" will have different primary weights. The secondary components typically correspond to accents, so "á" and

Listing 1.

The collation elements for the string Foo.

```
RuleBasedCollator c =
  (RuleBasedCollator)Collator.getInstance();
CollationElementIterator iter =
  c.getCollationElementIterator("Foo");

int element;
while ((element = iter.next()) !=
  CollationElementIterator.NULLORDER) {
        System.out.println("Collation element is: " +
          element);
        System.out.println(" primary:   " +
          Integer.toString(
                  CollationElementIterator.
                    primaryOrder(element), 16) );
        System.out.println(" secondary: " +
          Integer.toString(
                  CollationElementIterator.
                    secondaryOrder(element), 16) );
        System.out.println(" tertiary:  " + Integer.
          toString(
                  CollationElementIterator.
                    tertiaryOrder(element), 16) );
}
```

C++ REPORT

The Undisputed Authority on C++ Programming

What You'll Learn:

- How to Build Distributed CORBA Applications in C++
- Mastering Power UML Programming Techniques
- Tips on C++ Programming Efficiency
- How to Design in Patterns
- Demystifying Windows Programming Using VC++6.0 & MFC
- Large-Scale Software Design Techniques
- How to Avoid STL "Gotchas"
- Tips for Optimizing Plain and Multithreaded C++ Code
- Garbage Collection in C++
- COM Threading Models & the ATL
- How to Write Exception-Safe C++
- Programming Techniques for High-Performance C++

C++ REPORT
IMPROVING DESIGN WITH TEMPLATES
Tips for Measuring Your Program's Performance
C++ REPORT
MULTITHREADING
Conditions to Ensure STL Thread Safety
Portable Support for Thread Creation and Control

C++ Report is packed with new ideas, tips, tricks, shortcuts, and usable source code on every aspect of C++ development.

Subscribe today

and pay only $79 for 10 issues of C++ Report*

*Canadian and Mexican orders add $25. International orders add $49. All checks must be in U.S. dollars drawn on a U.S. bank.

www.sigs.com
email: subscriptions@sigs.com
Phone: 800-361-1279 (U.S. only)
615-377-3322 (Canada and Mexico)
+44 0 1858 435 302 (All other countries)

SIGS PUBLICATIONS

Component Strategies • C++ Report • Java™ Report
JOOP • Application Development Advisor (UK)
Objekt Spektrum (GER) • Java™ Spektrum (GER)

71 West 23rd Street, 3rd Floor, New York, NY 10010

"é" will have the same secondary weight, which is different from the secondary weight of "a". The tertiary components usually represent the case of a character, so "a" and "A" will have different tertiary weighting. (There is a fourth level, the normalized original string itself, which can be used for a final comparison, but you usually don't need to worry about this level.)

Table 1 uses simple, made-up numbers to illustrate the components of a collation element. In practice, the numbers are usually larger, because most collators have many more possible elements. To see what real collation orders look like, you can modify the last code example as follows. Listing 1 prints out the collation elements for the string Foo.

While this notion of three ordering levels seems complicated at first, it actually makes some tasks easier. If you want to do a case-insensitive comparison, you simply ignore the tertiary component of each collation element. If you also don't want to include accents, you can ignore the secondary component too. Collator handles this by allowing you to set different strength levels using the setStrength method and constants such as Collator.PRIMARY.

```
Collator c = Collator.getInstance();
c.setStrength(Collator.PRIMARY);
if (c.compare(string1, string2) == 0) {
    ...                                  // Strings matched
}
```

Listing 2.

Masking unwanted portions of the collation element.

```
// Return a mask for the part of the order we're interested in
static final int getMask(int weight) {
        switch (weight) {
                case Collator.PRIMARY:
                        return 0xFFFF0000;
                case Collator.SECONDARY:
                        return 0xFFFFFF00;
                default:
                        return 0xFFFFFFFF;
        }
}
boolean match(CollationElementIterator text,
                        CollationElementIterator pattern)
{
        int mask = getMask(weight);
        int done = CollationElementIterator.NULLORDER & mask;

        while (true) {
                int patternElement = pattern.next() & mask;
                int targetElement  = text.next()    & mask;

                if (patternElement == done) {
                    break;                      // End of pattern
                } else if (patternElement != targetElement) {
                    return false;               // Mismatch
                }
        }
        return true;                            // No mismatches
}
```

TEXT SEARCHING IN JDK 1.1

Now that you are familiar with the concepts behind collators and collation elements, we can put some of that knowledge to use. The same collation elements that RuleBasedCollator uses to perform string comparisons can be used to do string searching as well. The basic concept is quite simple: instead of searching through the characters in the string, we'll search through its collation elements.

Fast string-searching algorithms such as KMP and Boyer–Moore require the ability to back up or perform random access in the text being searched. Unfortunately, you can't do this with international text in JDK 1.1 because CollationElementIterator does not allow random access. It only has a next method and is lacking setOffset and previous. This means that Boyer–Moore searching cannot be implemented without a complicated buffering scheme that is very tricky to get right.

However, a traditional, brute-force string search is quite possible using the JDK 1.1 API. Essentially this comes down to comparing the search pattern against each individual character position in the text being searched. If the collation elements for the search pattern match the collation elements for that substring of text being searched, we've found a match. The outer loop looks like this:

```
// What to search for
String pattern = "for";
// Text being searched
String text = "Now is the time for all good men";

RuleBasedCollator c =
(RuleBasedCollator)collator.getInstance();
CollationElementIterator patIter =
    c.getCollationElementIterator(pattern);

for (int i = 0; i < text.length(); i++) {
        String substr = text.substring(i);
        CollationElementIterator textIter =
            c.getCollationElementIterator(substr);
        patIter.reset();
        if (match(patIter, textIter)) {
            // They matched!  Do something.
        }
}
```

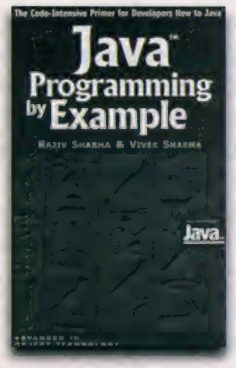
Of course, I left out the hard part, the function match that decides whether two sequences of collation elements are equivalent. A simple, naïve implementation would loop through both iterators and ensure that the elements they return are the same:

```
boolean match(CollationElementIterator text,
                CollationElementIterator pattern)
{
        while (true) {
        int patternElement = pattern.next();
```

Have a CUP of Java

Java Programming by Example

Rajiv Sharma** and **Vivek Sharma



Java Programming by Example introduces software developers to Java, the object-oriented programming language of choice for Internet development. Using nothing more than demonstrations and wry humor, the authors provide the information developers need to create quickly applets, applications, multi-file programs, animation, and GUIs.

Advances in Object Technology 17

1998 336 pp.
0-521-64442-9 Paperback $34.95

Component-Based Development for Enterprise System

Applying the Select Perspective

Paul Allen** and **Stuart Frost

The approach in this book offers a streamlined set of modeling techniques based on Unified Modeling Language (UML); uses a serviced-based architecture that provides an overall design philosophy for reusable software that addresses the needs of the business, not the technology; and is a clearly defined software process that provides guidance on how to employ the knowledge base of available techniques.

Managing Object Technology Series 12

1998 446 pp.
0-521-64999-4 Paperback $44.95

Building Object Applications that Work

Your Step-by-Step Handbook for Developing Robust Systems with Object Technology

Scott W. Ambler

Written by internationally-acclaimed trainer and developer Scott Ambler, this book is a must-have resource for designers, programmers and testers of today's OO applications.

Managing Object Technology Series 9

1998 506 pp.
0-521-64826-2 Paperback $39.95

Process Patterns

Building Large-Scale Systems Using Object Technology

Scott W. Ambler

Written by one of the best known object-oriented practitioners in the business, *Process Patterns* is based on proven, real-world techniques. Scott Ambler shows readers how to successfully deliver large-scale applications using object technology, and he carefully describes how one develops applications that are truly easy to maintain and to enhance.

Managing Object Technology Series 15

1998 385 pp.
0-521-64568-9 Hardback $44.95

More Process Patterns

Delivering Large-Scale Systems Using Object Technology

Scott W. Ambler

In this book, the author presents process patterns for the second half of the development lifecycle. He covers the Deliver phase and the Maintain and Support phase of large-scale, object-oriented system development. Each presented pattern is based on proven, real-world techniques and is geared toward medium to large-size organizations that need to develop software internally to support their main line of business.

Managing Object Technology Series 19

1999 c.384 pp.
0-521-65262-6 Hardback $44.95

SIGS BOOKS

Available in bookstores or from

CAMBRIDGE UNIVERSITY PRESS

40 West 20th Street,
New York, NY 10011-4211
Call toll-free 800-872-7423
Web site: http://www.cup.org
MasterCard/VISA accepted.
Prices subject to change.

Rethinking Smart Objects

Building Artifical Intelligence with Objects

Daniel W. Rasmus

Dan Rasmus has surveyed his own body of work and gathered the most important and relevant material from his contributions to the magazine. This is the first book to explain the integration of object technology and knowledge software development.

Advances in Object Technology 18

1998 244 pp.
0-521-64549-2 Paperback $32.95

The Patterns Handbook: Techniques, Strategies, and Applications

***Linda Rising**, Editor*

Rising has selected a large number of seminal articles and essays that illustrate the growing importance of patterns in application development. A partial list of well published experts includes James Coplien, Kent Beck, Grady Booch, Ralph Johnson, Robert Martin, Andrew Koenig, and John Vlissides.

SIGS Reference Library 13

1998 570 pp.
0-521-64818-1 Paperback $44.95

Developing Business Objects

***Andy Carmichael**, Editor*

Since each chapter of the book is firmly grounded in the real world of people, budgets, deadlines, successes and failures, it cuts through the hype to the real issues that affect the quality and productivity of every software engineering endeavor.

Managing Object Technology Series 11

1998 334 pp.
0-521-64825-4 Paperback $39.95

```
            int targetElement = text.next();
            if (patternElement = = CollationElementIterator.
             NULLORDER) {
            break; // End of the pattern
            } else if (patternElement != = targetElement) {
                return false;       // Mismatch
                    }
            }
            return true;    // No mismatches
    }
```

This will work, but only if you want to treat *any* difference, be it primary, secondary, or tertiary, as significant. In most applications, that is not enough; users will want an "Ignore Case" option, and possibly an "Ignore Case and Accents" option as well. Fortunately, this is fairly easy to do. Collator provides the constants PRIMARY, SECONDARY, and TERTIARY that you can use to represent the level of comparison you want, and CollationElementIterator provides methods to break down a collation order into its three components.

All we need to do is create a variable, e.g., weight, that stores the desired level of comparison. If weight is PRIMARY, we check only the primary component of each collation element. If weight is SECONDARY, we check both the primary and secondary components, and if it is TERTIARY we check all three. Fortunately, the values of these constants are in ascending numerical order, so we can use simple comparisons such as "if (weight > Collator.PRIMARY) ...".

To add this extra functionality we have to modify our match function a bit, but it is still fairly simple. Because the documentation for CollationElementIterator promises that "the first 16 bits of a collation order is its primary order; the next 8 bits is the secondary order and the last 8 bits is the tertiary order," we can simply mask away the portions of the collation element that we're not interested in (see Listing 2).

Listing 3.

Settings to re-use and compare the current pattern element, thus skipping the current text element.

```
boolean match(CollationElementIterator text,
                    CollationElementIterator pattern)
{
        int mask = getMask(weight);
        int done = CollationElementIterator.NULLORDER & mask;

        boolean getPattern = true, getTarget = true;
        int patternElement = 0, targetElement = 0;

        while (true) {
                if (getPattern) patternElement = pattern.next() & mask;
                if (getTarget) targetElement   = text.next()    & mask;
                getTarget = getPattern = true;  // By default get both

                if (patternElement == done) {
                break;                                  // End of pattern
                } else if (targetElement == 0) {
                getPattern = false;             // skip targetElement
                } else if (patternElement == 0) {
                getTarget = false;                      // skip patternElement
                } else if (targetElement != patternElement) {
                return false;                           // Mismatch
                }
        }
        return true;                                    // No mismatches
}
```

IGNORE THAT CHARACTER!

As you've probably guessed, something was left out again. There's one last complication: *ignorable* characters. In Unicode, an accented character can be represented in two different ways. The single Unicode character \u00e1 represents "á", but the pair of Unicode values \u0061\u0301 also represents "á". The \u0061 is just a lowercase "a", but the \u0301 is special. It's a "combining acute accent" that combines with the value before it to create the single "user-level" character "á".

These combining characters need special processing during a comparison. Because \u0301 is *only* an accent, its collation element has a secondary component but no primary or tertiary component. In a comparison that does not involve accents, we must ignore this element entirely. If we did not, we might end up comparing an accent in one string to a base letter in another string, which would give invalid results. For example, when doing an accent-insensitive comparison of "a\u0301b" and "ab", we want to skip the "\u0301" and go on to the next character; otherwise we'd compare "\u0301" and "b".

This logic is relatively straightforward, but it does make the code a bit more complicated (see Listing 3). The boolean variables getTarget and getPattern are used to decide whether to fetch the next collation element in the text and the pattern each time through the loop. Normally, both variables are true, but one or the other can be set to false if we want to skip an element. For example, setting getPattern to false and getTarget to true causes the current pattern element to be re-used and compared with the next text element, thus skipping the current text element.

This is about the best you can do with the JDK 1.1 API. You can add bells and whistles, such as searching backward though text by reversing the order of the outer loop that calls match, but you can't really implement a more efficient search without a lot of work.

IT'S BETTER IN JAVA 2

In Java 2, we are making quite a few improvements to the international classes in java.text and java.util. Among them are enhancements to CollationElementIterator that make it possible to

write faster and more powerful search routines. These changes are present in JDK 1.2 beta3 and now in Java 2. You can download the latest version from the Java Developer Connection Web site.

There are two major problems with the searching mechanism just outlined. First, it uses an inefficient algorithm that can, at worst, compare every character of the pattern against every character of the target, requiring a number of comparisons that is proportional to the size of the text being searched multiplied by the size of the pattern. (In practice, it's usually not quite that bad, however.) In computer science terms, if the size of the text is T and the size of the pattern is P, the search time is proportional to T·P, or is O(TP). Modern searching algorithms can do much better.

The second, and more obvious problem is that there is an awful lot of object creation going on in the last few examples. Every time through the outer loop we call substring, which creates a new String object, and then we create a new CollationElementIterator. This happens at every single position in the target string, which is woefully inefficient given the cost of object creation in Java (and in most other languages, for that matter).

This second problem is solved by two new CollationElementIterator methods that we have added in Java 2:

```
public void setOffset(int newOffset)
public int  getOffset()
```

These methods allow you to change the text position to which an iterator points and to retrieve its current position. With this flexibility, we can avoid all of the calls to substring and all of the extra iterator objects that we were creating before. The outer searching loop now looks like this:

```
String pat = "for";
// What to search for
String text = "Now is the time for all good men";
// Text to search in
RuleBasedCollator c =
(RuleBasedCollator)Collator.getInstance();
CollationElementIterator patIter =
c.getCollationElementIterator(pat);
CollationElementIterator targIter =
c.getCollationElementIterator(text);

for (int i = 0; i < text.length(); i++) {
        targIter.setOffset(i);
        patIter.reset();            // Or setOffset(0)
        if (match(patIter, targIter)) {
                // They matched!  Do something.
        }
}
```

This will be *much* faster because we're no longer creating new objects each time through

Too rushed to optimize?

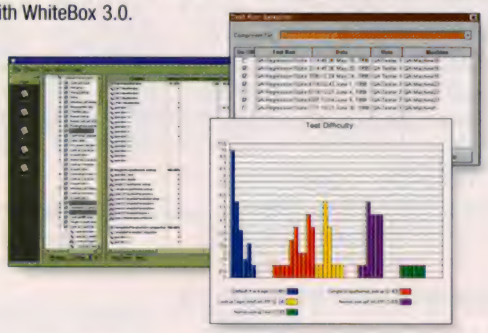
Think again...Introducing WhiteBox 3.0 test optimization and coverage all-in-one.

Getting your software out the door on time can put pressure on quality; that can mean shipping late or continually going over budget.

Test optimization is the solution. With WhiteBox 3.0, a complete code analysis and test optimization tool consisting of metrics and coverage. WhiteBox 3.0 allows developers and QA professionals to determine *where to test, how much to test,* and *when to stop testing,* greatly reducing testing costs. WhiteBox 3.0 optimizes your testing resources and provides metrics to meet your most important project quality needs.

WhiteBox 3.0 is specifically designed to not change the build process. It provides superior scalability, robust analysis with reporting and export capabilities, in addition to the ability to merge results throughout the project lifecycle and multiple test runs. WhiteBox 3.0 supports projects written in **Java**, as well as **C** and **C++**. And if a project is written in all three languages, it provides test optimization and coverage across all the source code. Developers will also appreciate how WhiteBox 3.0 conveniently integrates with Microsoft DevStudio and Symantec's Visual Cafe.

So don't let the time-to-market rush affect either software quality or project cost. Test smarter, test faster, and save money with WhiteBox 3.0.



WhiteBox VERSION 3.0

Reliable Software Technologies

21515 Ridgetop Circle, Suite 250, Sterling, VA 20166
703-404-9293 • Fax: 703-404-9295 • **whitebox@rstcorp.com**

www.rstcorp.com/whitebox
1-800-824-0022

the loop. The algorithm is still O(TP), but the overhead per iteration is considerably lower, so the running time will be a lot better.

OPTIMIZED SEARCHING

We have solved the easier of our two efficiency problems; now it's time for the hard one. As explained, the brute-force algorithm we're using is O(TP). String searching is a well-researched area, and there are algorithms that can do considerably better. Perhaps the best is the Boyer–Moore method, which is never worse than O(T+P) and in practice is often proportional to T/P. That's right: the size of the text *divided* by the size of the pattern. Rather than forcing us to examine characters in the text multiple times, this algorithm actually lets us *skip* characters.

Boyer–Moore can be a little bit tricky to explain, but once you "get" it, it seems almost too obvious. The trick is that instead of comparing the strings starting at the beginning of the pattern, you compare them starting at the end. If the characters don't match, we still gleaned a bit of useful information: we now know what character occurs at that position in the text being searched. Often, we can take advantage of that information to skip several characters in the target text rather than simply sliding the pattern along by one position and trying again.

An example will make this more clear. Imagine that you're searching for the word "string" inside the phrase "silly spring string." To start off, you line up the beginning of the pattern with the beginning of the target, but you start comparing at the end, like so:

```
silly spring string
string
```

We compare the **g** in the pattern with a space character in the target, and there's no match. So far, there's nothing special. However, we know something else as well: the character at index 5 in the target is a space, and there are no space characters anywhere in the pattern we're searching for. Combining these two facts, we can slide the pattern over by six characters and try again:

```
silly spring string
      string
```

This time, there is a match between the pattern and the text. Because we're going backwards, we now compare the **n**, **i**, and **r** and find that they match too. However, the p and the **t** do not. We know that there is not a **p** anywhere in the pattern, so we can slide it over again:

```
silly spring string
         string
```

This time, we see an **s** in the text. It's not a match, but we do know that there is an **s** at the beginning of the pattern. Therefore, we slide the pattern over five spaces. Now we have the following, which gives us a match.

```
silly spring string
              string
```

To implement this efficiently, you need to have a table that, for each possible character in the text, tells you how far from the end of the pattern that character first occurs. This is the distance by which the pattern can be shifted when that particular character is seen in the input. For the above example, it would look like Table 2.

This table can be computed once, before the search is started, by making a single pass through the pattern. After that, it can be used each time we search for that pattern, leading to a huge performance win.

BOYER–MOORE VS. UNICODE

But wait! At the very beginning of this article, I said that this kind of algorithm doesn't work

Listing 4.

Calculating Boyer–Moore shift distances with Java 2.

```
// Member variables for storing precomputed pattern data
private int   patLen;
private int[] patElem;
private int[] shifts;

// Map a collation element to an array index
int hash(int order) {
        return CollationElementIterator.primaryOrder(order) % 256;
}

// Initialize the Boyer-Moore shift tables
void initialize(RuleBasedCollator c, String pat)
{
        // First find out how many elements we're dealing with
        patLen = 0;
        CollationElementIterator iter = c.getCollationElementIterator(pat);
        while (iter.next() != CollationElementIterator.NULLORDER)
                patLen++;

        // Allocate space to store the pattern elements and the shift tables
        patElem = new int[patLen];
                shifts = new int[256];

        // Elements not found in the pattern get the maximum shift distance
        for (int i = 0; i < 256; i++) {
                shifts[i] = patLen;
        }

        // Now compute the shift distances for the elements in the pattern.
        // While we're at it, save the elements themselves for quick access.
        // The "-1" is in the calculation because Java indices are 0-based.
        iter.reset();
        for (int i = 0; i < patLen; i++) {
                patElem[i] = iter.next();
                int index = hash(patElem[i]);
                shifts[index] = Math.min(shifts[index], patLen - i - 1);
        }
}
```

well with Unicode because it has 65,535 possible character values, which would make the table too large. Actually, it's worse, because we're concerned with collation elements, which are 32-bit integers, not with the Unicode values themselves. That's true, but (of course) there's another trick...

First, consider what happens when a letter occurs twice in the search pattern. There are two possible shift distances for that letter, one for each occurrence. To make the Boyer–Moore algorithm work, we always want to enter the smaller of the two shift distances in the table. If we used the larger one, we might shift the pattern too far and miss a match. In a sense, the shift table is not required to be perfectly accurate, and conservative estimates of shift distances are OK. As long as we don't shift the pattern too far, we're fine.

This realization leads to a simple technique for applying the algorithm to Java collation elements: simply map all possible elements down to a much smaller set of shift table indices (say, 256). If two or more elements in your pattern happen to collide and end up with the same index, it's not a problem as long as you enter the smaller of the shift distances in the table.

A simple way to map the collation elements to 256 values is to use the low byte of the element's primary component. There are other approaches that will lead to a better distribution of values throughout the shift table and will thus give slightly better performance, but in practice this approach is usually good enough.

To implement Boyer–Moore searching with Java 2, we first need to construct a shift table that tells us how far to shift the pattern when a particular collation element is seen in the text (see Listing 4). The hash function is used to map from a 32-bit collation order down to an index in the 256-element array.

Once we have the tables, the search routine is straightforward. It uses another new method: `CollationElementIterator.previous`. Also note that there is no longer an outer loop that calls a separate match method, because that only worked well when we were marching through the text one character at a time. Now that we can skip ahead an arbitrary distance through the text, it is easier to combine all of the logic into one method (see Listing 5).

There you have it: a way to do fast, linear-time, international-friendly string searching in Java.

THE REAL STUFF

I hope I have given you a good idea of how you can use collators to add language-sensitive sorting and searching to your own Java applications. It is not that hard and the benefits can be enormous, because global markets are becoming increasingly more important to the computer industry. For example, according to IBM's first quarter 1998 financial statement, more than half of IBM's revenue came from outside North America. Using the international features of the JDK can help you begin to tap into this huge market.

The code examples listed in this article were intended primarily for their educational

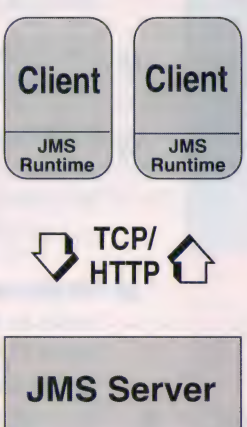
Table 2. Boyer–Moore Shift Distances.

INPUT CHARACTER	DISTANCE
s	5
t	4
r	3
i	2
n	1
g	0
others:	6

Goodbye proprietary messaging! Introducing the...

Fiorano JMS Server

Dramatically reduces development time for complex network applications.



- First 100% pure Java implementation of JMS (Java Message Service) API's. Includes **Pub/Sub**, **PTP (Message Queues)**, **Transactions**, **Durable Subscribers** and more.
- **Lightweight**, **efficient** and **robust** environment; thin runtime to access JMS Services.
- **Integrates with popular Application Servers**.
- Efficient Server-to-Server message routing allows **massive scalability.**

Download a **30-day free evaluation** at **www.fiorano.com**

Fiorano

Fiorano Software, Inc.
236 N. Santa Cruz Ave., Suite 213 • Los Gatos, CA 95030
Tel 408.354.3210 • Fax 408.354.0846
e-mail: info@fiorano.com • http://www.fiorano.com

Object E·X·P·O

THE NATIONAL CONFERENCE & EXPOSITION

SIGS EXPO for JAVA™

Conference
March 28–April 1, 1999

Exhibition
March 30–31, 1999

New York Marriott
World Trade Center, New York, NY

Increasing Productivity with object, Java and component-based development.

Shorten development cycles. Increase reuse. Reduce project risk. Improve software quality. Get the hard facts on how to attain these project goals by attending the leading technical forum for serious application development. Select from 90+ hand-picked, no-nonsense sessions over 5 information-packed days. Filled with usable techniques, practical strategies and proven solutions needed for building large-scale distributed systems.

What's New in 1999?

1. We're moving to the heart of Wall Street at spectacular World trade Center – the international capital of the world. Convenient and easily accessible from the tri-state area and around the world.
2. **A fresh new technical program with scores of new classes on today's hottest topics**–UML Process, Catalysis, CBD, EJB, Agents, OMG's Object Management Architecture, Java 2, CORBA 3.0, EJB App Servers, Use Cases, MTS, Embedded Java, design patterns, XML, Swing and much more. The 1999 technical program will be objective, product neutral and fast paced–as the well-respected pages of JOOP, Java Report™ and Component Strategies "come to life".
3. **Pre– and Post–conference tutorials** scheduled on Sunday and Thursday. These sessions will compliment and enhance the skills you will develop during the 3 day Conference.
4. **Night School.** Why stop when the sun goes down? Night sessions can be taken individually or as part of the day program (14 sessions in all).
5. **Learn from industry experts and sought-after consultants.** They've already taught thousands of software developers to be more productive and will do the same for you.The two day exhibition will be packed with 50+ leading vendors showcasing the latest releases.
6. **Introducing the "Component Users Forum."** Free to all attendees, this special event will feature 16 distincts, 50 minute vendor sponsored case studies throughout Tuesday and Wednesday. The two day summit will cater to those seeking and implementing component-based systems.

Component Users Forum

Celebrating our 11th successful year.

For more information, contact us at:

SIGS Conferences
71 West 23rd Street
New York, NY 10010

T: (212) 242-7447
F: (212) 242-7578
E: conferences@sigs.com
URL: www.sigs.com

Presented by:

SIGS CONFERENCES

Sponsored by:

JOOP The Journal of Object-Oriented Programming

Component STRATEGIES

Java REPORT THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT

Future of Application Development

The must alter educational event for anyone looking to thoroughly understand the "ins and outs" of large scale development in a net centric environment. You will discover the complete world of object technology and see how to make it work for you.

12 Hi-Powered Tracks–Covering the Spectrum of "Hot" Application Development Topics

- Advanced C++
- Distributed Objects
- Object Technology Fundamentals
- Software Development
- Components
- Programming Fundamentals
- Java Programming Tips
- Databases
- Patterns/ Framework
- Windows Programming
- Distributed Java
- Java for Gurus

What You Will Learn By Track – 90+ Intensive Sessions in All! (partial list)

❶ Advanced C++

- Writing Exception Safe C++
- Uses and Abuses of Inheritance
- Tips for Optimizing "Plain" and "Multi-threaded" C++ Code
- Mastering Design Patterns in Java

❷ Distributed Objects

- CORBA and RMI Programming Models
- Implementing and Using the CORBA Portable Object Adapter Framework
- Developing a Component Framework with CORBA
- Agents- the next components
- Understanding COM architecture
- Using COM to Wrap the Enterprise: What works and what doesn't
- Distributed Component-Based Event Monitoring and Debugging

❸ Object Technology Fundamentals

- OO Project Management
- Choosing DCOM or CORBA
- Object Transition Management
- Building Distributed OO Applications whit Java Servlets
- A New Approach to 100% Java Client/Server Application Development
- Practical Strategies for Migrating to CBD
- DCOM Security

❹ Software Development

- Requirements Engineering
- Effectively Using UML 1.1
- Analysis and Design Issues of Multi-tier Architectures
- Where Distributed Objects Meet Transaction Monitors
- Advanced Use Case Analysis
- A Comparison of EJB Application Servers
- Enhancing your Projects with Business Component Analysis

❺ Components

- Server Side Components with CORBA
- Building Business Component Architectures with Enterprise JavaBeans and CORBA
- Rapid Application Development Strategies
- Modeling COM with UML

❻ Patterns/ Framework

- Applying Patterns in Java
- Frameworks and Patterns for Distributed Systems
- Process Patterns: Reusable Approaches to OO Development

❼ Programming Fundamentals

- Applets to Servlets: Java Client/Server Communication on the Real World Web
- Using the Standard Template Library in C++
- Building Language Independent Components with Active X
- Relational Data Binding with SWING

❽ Windows Programming

- Interface-Based Programming in MFC
- Layout Management in MFC
- Microsoft Transaction Server
- Intro to Java RMI

❾ Databases

- Mapping Objects to Relational Databases
- Integrating Java whit Relational Data
- Developing 3-tier Database Applications with Java Servlets
- Successfully Architecting, Building and Deploying Ecommerce Systems

❿ Java for Gurus

- Advanced Java Interface Design
- Advanced SWING Component Integration
- Creating Java Wrappers Over Legacy Client/Server Systems
- Java Debugging and Profiling Techniques
- Mastering Object Serialization and RMI
- Multithreaded Exception Handling in Java
- Jini and Java Spaces
- Refactoring in Java
- Programming the Java Layout Manager

⓫ Java Programming Tips

- Implementing the XML in Java
- Developing Multi-tier Applications Using the Servlet API
- Building Server Objects in Java
- Java in Embedded Systems
- Building a Multi-threaded Server in Java
- Really Understanding Java Threads
- Developing JavaBeans for Various Integrated Development Environments
- Integrating Java and Legacy Systems
- Developing with JFC
- Java 2 Preview
- Dynamic Page Compilation

⓬ Distributed Java

- Designing Java Applications with Distributed Systems
- Java, JavaBeans and EJB: The Return of Transaction
- Processing and What it means to Distributed Systems
- Developing Enterprise Solutions with EJB
- Building Java Objects with CORBA
- Introducing Java Servlets
- Using Java, CORBA and XML for a Distributed Object Architecture
- Building Transactional Components with EJB
- Building Large Scale Applications with EJB
- Bean Soup

MEDIA SPONSORS

Java Report
C++ Report
dejanews
JOOP
Dr. Dobb's Journal
MSJ
webreference
WEB Developer.com
EARTHWEB
Application Development Trends
CUTTER CONSORTIUM
C/C++ Users Journal
developer.com
INFO WORLD
PCWEEK
Component Strategies
Windows Developer's Journal
Application Development
CLIENT SERVER NEWS
Software Development

value, but they do work. However, for clarity I have ignored a few remaining issues to make the code easier to understand. Expanding ("ä" to "ae") characters in the pattern are the chief difficulty. If the shorter, non-expanded version of the character occurs in the text being searched, you can end up shifting too far and missing a possible match. However, it is not too hard to compensate for this by using the getMaxExpansion method (also new in Java 2) of CollationElementIterator to decrease the shift distances when expanded characters are seen in the pattern.

The other major feature I have left out is the ability to tell how much of the text matched the pattern. All of the code examples search for location in the text where a match starts, but they do not return the length of the text that matched. You would need to know this if you were writing a search and replace function in an editor, for example. In Java 2, it is easy to tell where the match ends: just call the iterator's getOffset method. With the JDK 1.1 API, it is harder; you basically have to resort to the brute-force technique of comparing longer and longer substrings of the text to the pattern and stopping when the collator says that they are equal.

If you want to see real, production-quality code that uses the same algorithms and solves these last few problems, visit the IBM alphaWorks Web site and download our fully-functional StringSearch class based on the JDK collation API. It supports case- and accent-insensitive searches, backward searches, and "whole word" searching among other features. alphaWorks also contains several other Java internationalization utilities that you might find useful, as well as a large number of JavaBeans, utility classes, and even a free XML parser written in Java.

Listing 5.

Boyer–Moore searching with Java 2.

```
public int find(String text, String pattern)
{
        RuleBasedCollator coll = (RuleBasedCollator)Collator.getInstance();
        CollationElementIterator targIter =
                                        coll.getCollationElementIterator(text);

        // build the shift table and the constants we need
        initialize(coll, pattern);
        int mask = getMask(weight);
        int done = CollationElementIterator.NULLORDER & mask;

        // Start at the text position corresponding to the end of the pattern
        int textIndex = pattern.length();

        while (textIndex <= text.length()) {
                boolean getPattern = true, getTarget = true;
                int targetElement=0, patternElement=0;

                iter.setOffset(textIndex);
                int patIndex = pattern.length();

                // Iterate backward until we hit the beginning of the pattern
                while (patIndex > 0)
                {
                        if (getTarget)  targetElement  = targIter.previous() & mask;
                        if (getPattern) patternElement = patElem[--patIndex] & mask;
                        getTarget = getPattern = true;

                        if (targetElement == 0) {
                            getPattern = false;             // skip targetElement
                        } else if (patternElement == 0) {
                            getTarget = false;              // skip patternElement
                        } else if (targetElement != patternElement) {
                                // There is a mismatch at this position.  Decide how far
                                // over to shift the pattern, then try again.
                                textIndex = iter.getOffset() +
                                                shifts[hash(targetElement)];
                                break;
                        }
                }
                if (patIndex == 0) {
                    // We made it back to the beginning of the pattern,
                    // which means we matched it all.  Return the location.
                    return targIter.getOffset();
                }
                // Otherwise, we're here because of a mismatch, so keep going....
        }
        return -1;          // No match.
}
```

ACKNOWLEDGMENTS

The patented technique for applying the Boyer–Moore search algorithm to collation elements was developed by Dr. Mark Davis of IBM. Kathleen Wilson, the manager of the text and international groups at IBM's Center for Java Technology in Silicon Valley, was very indulgent of the time I spent working on this article and the accompanying code. I would also like to thank Mark, Kathleen, Michael Pogue, John Raley, and Helena Shih for reviewing drafts of this article.

RECOMMENDED RESOURCES

Sedgewick, R., *Algorithms in C, 3rd ed.*, Addison-Wesley, 1997—Has a good overview of the Boyer–Moore search algorithm as applied to small character sets. He also covers a number of other search algorithms.

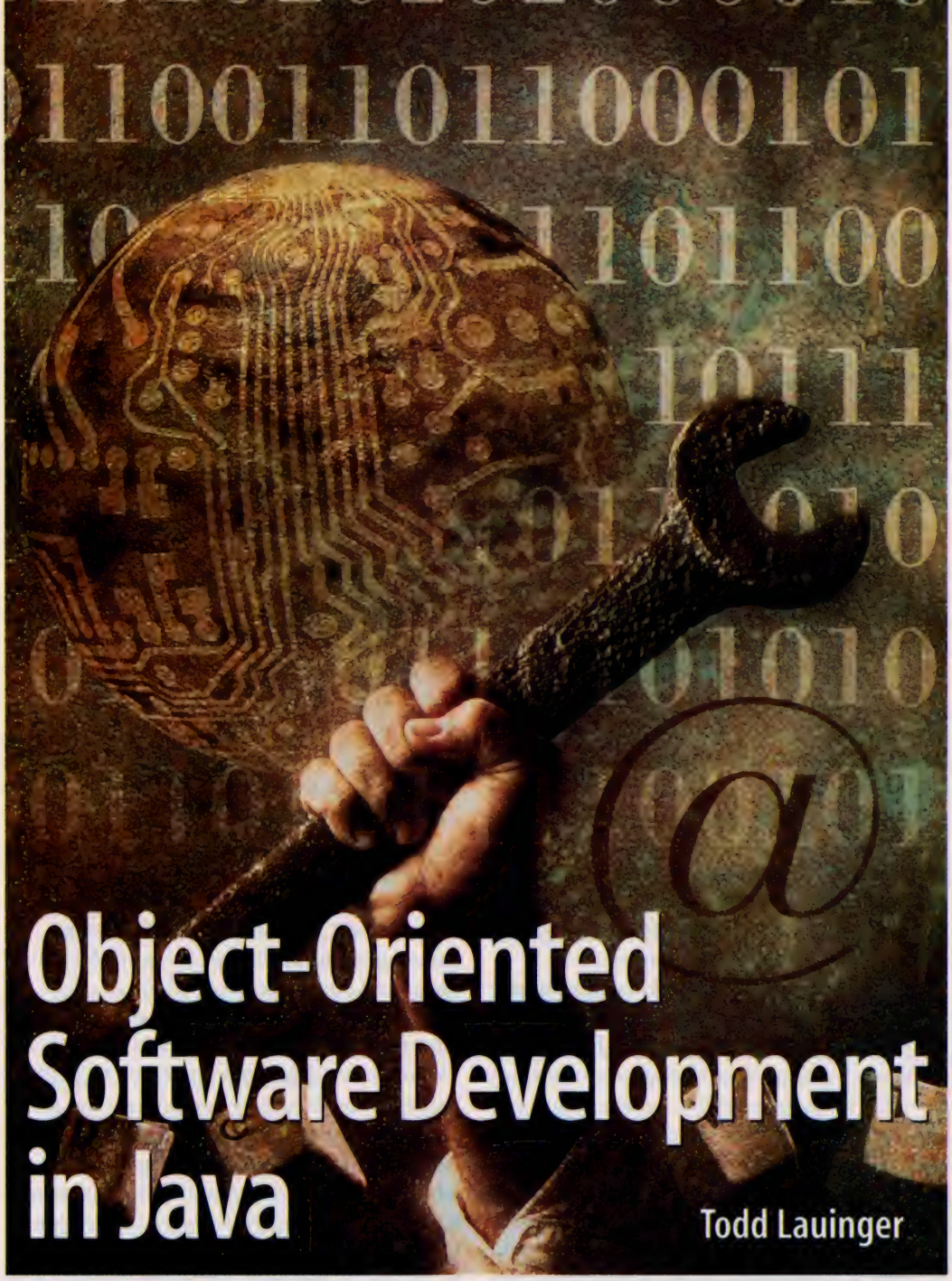
Chan, P., R. Lee, and D. Kramer, *The Java Class Libraries, 2nd ed., Vol. 1*, Addison-Wesley, 1998—Has a nice description of CollationElementIterator.

Making your Java/C++/C Applications Global, at www.ibm.com/java/education/international-unicode/unicode1.html—Is a good overview of some of the issues involved in writing global applications.

The Unicode Consortium's *The Unicode Standard, Version 2.0*, Addison-Wesley, 1996—Has lots of useful information on processing international text. In particular, section 5.15 covers sorting and searching.

The Unicode Consortium Web site at www.unicode.org has lots of useful information. If you want an even more detailed explanation of Uni-

continued on page 77



Gary and Vivian Chapman/Image Bank

Balancing Java tool knowledge with object technique

I BELIEVE YOU can get a good look at the software industry in general by looking at the conferences people attend. Java is, of course, smoking (or should I say steaming)! This year's JavaOne conference had around 14,000 attendees. Last year the conference pulled in 10,000 attendees, and the year prior 6,000 attendees. Microsoft isn't doing too badly, either. I attended the Microsoft Professional Developers conference in September 1997, and they had 10,800 attendees drooling over all the free CD-ROMs of software given away. This year's conference in Denver was down to 6,200 attendees, but still a significant turnout compared with most conferences.

As an object-oriented (OO) software developer, I like to balance my knowledge of tools with technique. The first conference I attended was the 1996 IBM International Conference on Object Technology. I believe there were somewhere in the neighborhood of 1,000 attendees there. In 1997, attendance had 600-700 participants. In 1998, IBM merged this conference with another. Hmm, do I see a trend here? SIGS merged its Object Expo conference with SIGS EXPO for Java years ago. OOPSLA, the premier object conference of the year, is holding its own at somewhere in the neighborhood of 2,300 participants per year. Still, this is barely one quarter the size of the

major tools-related conferences just mentioned.

I am truly at a loss with these statistics. Java is a fully OO language. As the demand for Java tools and development environments grow, so should the demand for good OO technique. Perhaps the Java conferences are teaching technique as well as tools? If JavaOne is any indication, this is clearly not the case. A colleague of mine found only one session on object technique, that being a presentation on patterns. Of course, JavaOne is being put on by a tool vendor, what should we expect? Maybe we need another indicator of the industry, like books published.

The bookstores I browsed have rows of bookshelves dedicated to Java programming (and healthy sections on the Microsoft tools as well). Some of the better ones have a small section that contains "advanced programming" topics like OO programming. The last straw that spurred me to create this article was a rather straightforward statement from a book publisher. I submitted a book proposal on mission-critical OO software development and it did not even get reviewed. Their response was "...we don't plan to make books on software development part of our upcoming publishing program. The lack of newsgroup interest is one reason, along with the fact that we're focusing on more application-specific programming books."

Now I am clearly convinced that the industry is ignoring technique in favor of tools. Giving your developers a Java toolset and a host of APIs without a foundation of OO software development is like giving a soldier a machine gun and not teaching them how to shoot. They may shoot very fast, but how often will they hit the target? And what will they hit *instead* of the target?

My purpose here is to wake up the industry from its deep object technique slumber. I will explain, at a high level, the core techniques that are widely accepted to produce quality OO software. These techniques include the following:

- Use Cases
- CRC Cards
- Responsibility-Driven Design
- Interaction Diagrams
- Class Diagrams
- Patterns

Each technique's explanation will contain one or more definitive references on the subject. With this information, you can wade through the tool overload in the industry and become versed in technique before writing even one line of production Java code.

USE CASES

Use Cases were invented by Ivar Jacobson and published in his book *Object-Oriented Software Engineering*[1]. Use Cases serve to expose the set of system capabilities (and users of those capabilities) that the system must provide. Use Cases get you out of the developer mentality of "which API will I use" and put you into the mentality of the user of the system. The best-designed software systems do what the user needs to do, nothing less, and nothing more. Use Cases precisely tell what the system must do, without getting into the dirty details of how the system will go about doing it. Systems designed without Use Cases tend to quickly loose focus of what the system is supposed to do for the user.

Use Cases drive the rest of the software development process. They are prioritized and scheduled for iterative development in a project plan. All other system development activities should be traceable to the Use Cases, as they describe the capabilities the system must provide. Software development driven by Use Cases lead to the production of excellent quality software systems.

CRC CARDS

Kent Beck and Ward Cunningham devised the concept of CRC Cards for OO design. The concepts were published in depth by David Bellin and Susan Suchman Simone in *The CRC Card Book*[2] and also in an earlier work, *Using CRC Cards: An Informal Approach to Object-Oriented Development*[3], by Nancy M. Wilkinson. CRC Cards help developers get out of the coding mentality and into the object analysis arena. CRC stands for Class, Responsibility, Collaboration, the three main elements documented on a CRC Card. These elements are listed on the front of a 3x5-index card—yes, three inches by five inches only. Any class that you document that has more responsibilities than can be written, one per line, on a 3x5 card is not correctly designed. You are not taking advantage of the object paradigm to create simple yet intelligent objects. Well-designed objects work together in a variety of ways to solve lots of different problems. For each responsibility listed on a CRC Card, you provide a set of zero or more collaborating classes. Collaborators are classes whose responsibilities are needed to help fulfill your class's responsibility.

CRC Cards have many advantages over other techniques. First of all, they do not require a sophisticated notation or CASE tool. Second, they are quick and easy to create, destroy, and change. Any design at this early stage of devel-

opment must be easily changed, as your first object design is almost never your best. Third, the cards make it easy to role-play the classes and do design walk-throughs with your peers. The best object designs have objects that form natural abstractions of the problem domain. This role playing aspect assures that your classes form good abstractions, otherwise you will catch the awkwardness during role-play.

RESPONSIBILITY-DRIVEN DESIGN

CRC Cards are based on the design technique of structuring classes to fulfill responsibilities. This approach leads to classes that excel in terms of reuse, simplicity, power, extensibility, and maintainability. Contrast this with data-driven or procedural-driven approaches for designing classes. Data-driven designs have lots of container classes, which are classes that simply contain and provide accessors to data. These classes are then controlled by other very complicated classes that do the work of the system. The control classes become very large and very difficult to change or extend. Procedural-driven approaches are worse, as the developer designs the system to solve a particular problem, often without regard to data abstraction. When business requirements change, the entire procedural design often needs to be scrapped and rewritten from scratch.

Responsibility-driven design is best described in Rebecca Wirfs-Brock's book, *Designing Object-Oriented Software*[4].

INTERACTION DIAGRAMS

Interaction Diagrams display, in a graphical manner, the sequence of events that take place as an object attempts to fulfill a non-trivial responsibility. The sequence is scenario-specific and drawn according to a timeline of method calls and returns. The scenario documented in an Interaction Diagram takes into account a specific set of conditions under which a responsibility is to be fulfilled. This is in contrast with a Use Case, which describes a system capability under all possible scenarios. Interaction Diagrams are described in almost every recent OO software development book and are documented quite thoroughly in Jacobson[1].

Interaction Diagrams have two main advantages. The first is the ability to visually diagram a collaboration scenario. This diagram rapidly points out the misallocation of responsibilities in the system. Objects that have too much responsibility "hog" the diagram, calling the accessors of lots of other objects and performing complicated logic on a combination of the data retrieved from those objects. Objects that have too little responsibility act as data sources for those with too much responsibility. A good design frequently produces Interaction Diagrams where message calls and returns are distributed evenly among the members of the diagrams. The second advantage of Interaction Diagrams is the ability to detect parallel processing problems like race conditions and deadlocks.

CLASS DIAGRAMS

As I get more comfortable with my CRC Card design, validated by role-playing sessions and Interaction Diagrams, I create a layout of my classes in a Class Diagram. The Class Diagram allows you to add a number of vital elements to your design, particularly relationships and cardinality. Relationships and the cardinality on those relationships help solidify the business rules that your classes must obey later in code. Class Diagrams also allow you to create a visual representation of how your classes fit together as a whole, which is a bit difficult to see from CRC Cards and Interaction diagrams.

Class Diagrams have been prevalent in OO software development for a long time. The best recent work I have found, taking into account the popular UML notation, is Martin Fowler's book, *UML Distilled*[5]. Practically any good OO software development text will have excellent information on this technique as well.

PATTERNS

The concept of software development patterns actually comes from work done by a building architect (as opposed to a software architect) named Christopher Alexander. In his work, he tried to identify timeless ways of designing buildings and, in fact, entire societies that people enjoyed living in and being part of. He then devised a format to express these patterns and a language to relate the patterns to each other. Object-oriented software developers adopted this notion of patterns to software development, creating all kinds of patterns and even a few pattern languages. The principle types of software patterns are architectural, analysis, accountability, and design patterns and software language-specific idioms. Of these types, the analysis and design patterns are of particular interest to those new to the pattern concept. Two excellent references to start with are Martin Fowler's *Analysis Patterns: Reusable Object Models*[6] and the "Gang of Four" book, *Design Patterns: Elements of Reusable Object-Oriented Software*[7]

Patterns capture the knowledge of experienced software developers in a form that can be

continued on page 73

Supercharged Strings

Steve Ball
steve@effectivejava.com

Steve Ball is an independent contractor specializing in C++/Java/UNIX development. He currently lectures in Object-Oriented Design and C++ Programming at Auckland Institute of Technology.

STRINGS HAVE BEEN awarded a special status in the Java language. The String class is the only reference type that can be instantiated by a compile-time literal and the only one to have its own operators (string concatenation, + and +=). Despite its privileged rank, there is one thing you can't do with a string that you can do with an instance of almost any other class: you can't change its value.

The immutability of strings has a number of happy consequences for the Java developer. I'll demonstrate how taking advantage of their special properties can improve the efficiency of code that works with string objects. I'll also show what the Java Language Specification has to say about supercharging string operations and how not everything it says is what it seems.

MAXIM 11: USE STRING OBJECTS IN PREFERENCE TO CHARACTER ARRAYS

A string literal is a sequence of zero or more Unicode characters enclosed in double-quotes. The compiler replaces each literal with a reference to an instance of the String class that contains the quoted characters.

Because a string once constructed cannot be modified, multiple identical occurrences of a string literal can share the same String instance without the risk of the object being altered through one reference and affecting the other. The Java virtual machine (JVM) exploits this opportunity, replacing multiple occurrences of the same string literal with references to the same unique String instance.

```
String s1 = "twine";
String s2 = "twine";
boolean shared = (s1 == s2);    // true
```

Concatenation between string literals is performed at compile-time, so these two strings will also refer to the same instance:

```
String s1 = "twine";
String s2 = "tw" + "ine";
boolean shared = (s1 == s2);    // true
```

This means that there is no runtime overhead incurred if you split a string literal across multiple lines to prevent it from exceeding the line length of your editor:

```
String message = "Warning: The system will be " +
    "shutting down in one minute. Please exit " +
    "your programs and log off";
```

These strings will be concatenated to form a single String instance when the class is compiled, incurring a minuscule time penalty then, but none when the program is subsequently executed.

The Java Language Specification 1.0 (JLS) (§3.10.5) states emphatically that the JVM merges (or, to use its phraseology, *interns*) string literals across different classes and packages. When a class is loaded, its string literals are merged with those already loaded. String literals that match any of the ones that have already been loaded are replaced with references to existing String instances. The remaining string literals require new String instances to be constructed, which may themselves become shared when subsequent classes are loaded. So even if s1 and s2 had been variables of different classes from different packages, they would still refer to the same object.

The JLS further states that "Strings computed by constant expressions (§15.27) are computed at compile time and then treated as if they were literals." So these two strings should both refer to the same unique instance also:

```
String s1 = "Route 66";
String s2 = "Route " + 66;
boolean shared = (s1 == s2);    // true
```

That is the theory anyway. In practice, the Java

Table 1. Compiler support for string interning rules.

	JDK 1.0	JDK 1.1	JAVA 2	VISUAL J++ 1.1
String literals from same class merged †	✓*	✓	✓	✓
String literals from different classes merged †	×	✓	✓	✓
String literals from different packages merged †	×	✓	✓	✓
String literals concatenated at compile time ‡	✓	✓	✓	×
String based on constant (non-string) expressions concatenated at compile time ‡	×	×	✓	×

*The JDK 1.0.2 runtime does not merge identical string literals occurring in the same class under all circumstances. See the Effective Java Web page for further details.

† Function of the Java runtime environment (JVM).

‡ Function of the Java compiler.

runtime may not go to quite these lengths to ensure that all possible duplication is eliminated, making the number of actual String instantiations as few as possible. Compilers and runtimes vary in their levels of compliance with the JLS string interning rules. Table 1 shows which compilers and runtimes obey which rules.

Object construction is one of the most expensive atomic operations in Java (McManus, A. & J. Hunt, "The Need for Speed," *Java Report*, Vol. 3, No. 5, 1998). Any mechanism that reduces the number of object instances constructed is worth taking advantage of. By using String objects instead of character arrays for constant string values you allow the runtime system not only to save space in memory (reducing the load on the virtual memory system) but also time in loading the class.

The elimination of redundant String instantiations is not the only benefit of string immutability though. It also allows the String class's own methods to be implemented much more efficiently than would otherwise be possible.

EFFICIENT STRING METHODS

If a String object can never be modified, it follows that sub-ranges of its character sequence must also be constant. This permits a number of the String class's methods to be implemented very efficiently. The substring method is a good example. The length of time taken to perform the operation does not depend on the length of the extracted sub-string. It executes in *constant time*.

```
String s = "A very long string... (>100 chars)";
// extract the 20th through 99th characters
String ss = s.substring(20, 100);
```

The String class is implemented using private instance variables to hold a char array for the characters of the string, the string's length, and an integer offset indicating the string's starting position in the char array. The substring method may be given a compact implementation that simply constructs a new String object that shares the same char array but with different values for the length and offset.

```
private String(char[] chars, int length, int offset) {
    this.chars = chars;
    this.length = length;
    this.offset = offset;
}
public String substring(int begin, int end) {
    return new String(this.chars, end - begin,
        this.offset + begin);
}
```

This is an advantage that Java has over other languages, like C++, whose string objects are mutable. No similarly efficient implementation is possible in C++ for the substr member function of the STL string class, which is forced to fabricate a private copy of the character subrange to prevent any of these characters from being later modified via the originating string. It operates in *linear time* (with respect to the length of the sub-string), unlike Java's substring method, which executes in constant time.

The same penalty is imposed (in Java) with character arrays. Extracting a sub-string, as another character array, also has linear time performance. The characters must be copied out using System.arraycopy into a separate array—the length of time taken is a function of the number of characters copied.

This is another good reason to prefer String objects to character arrays: to take advantage of the String class's uniquely efficient methods.

MAXIM 12: ELIMINATE TEMPORARY STRINGS AND SAVE TIME

The examples so far have dealt with constant string values. For storing modifiable strings, Java provides the StringBuffer class. String

ADVERTISER INDEX

Advertiser	Page	URL	e-mail
4th Pass	CV 2	www.4thpass.com	rwolf@4th pass.com
Ad Astra Engineering	36	www.JumpingBeans.com	crygaard@AdAstraEng.com
Advanced Software Technology	33	www.advancedsw.com	sales@advancedsw.com
BEA	2	www.beasys.com	
Cambridge University Press	51	www.cup.org	marketing@cup.org
C++ Report	49	www.sigs.com	subscriptions@sigs.com
IBM	5	www.software.ibm.com/websoftware	
ILOG	9	www.ilog.com	jviews@ilog.com
InstallShield	CV 3	www.installshield.com	william@installshield.com
Intel	21	http://developer.intel.com/vtune	
Intuitive	31	www.optimizeit.com	optimizeit@intuisys.com
KL Group	CV 4	www.klg.com	info@klg.com
KL Group	39	www.klg.com	info@klg.com
Mecklermedia	41	www.internet.com	siwprogram@mecklermedia.com
Microsoft	1	www.msdn.microsoft.com/visualj	
Modena	55	www.fiorano.com	info@fiorano.com
Numega	29	www.numega.com	carlm@numega.com
Object People	47	www.objectpeople.com	rausten@objectpeople.com
Object Expo	56-57	www.sigs.com	conferences@sigs.com
OOPSLA 99	46	www.acm.org/sigplan/oopsla	oopsla99@acm.org
Oracle	13	www.oracle.com/info/27	
Parasoft	43	www.parasoft.com/jtest	joe@parasoft.com
Petronio	35	www.petronio.com	info@petronio.com
Progress	23	http://apptivity.progress.com/rocket1	
Rational	15	www.rational.com/mvs/vismod/	info@rational.com
Rogue Wave	19	http://www.roguewave.com	shryock@roguewave.com
RST	53	www.rstcorp.com	ambruno@rstcorp.com
Tidestone Technologies	25	www.tidestone.com	sales@tidestone.com
XML World	67	www.sigs.com	conferences@sigs.com
Zero G	7	www.zerog.com	info@zerog.com

Java™ REPORT
THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT • www.javareport.com

BPA INTERNATIONAL

Group Sales Director
Miles E. Silverman
71 W. 23rd St., New York, NY 10010
• phone (212)242-7447 • fax (212)242-7574
• email msilverman@sigs.com

Advertising Sales Manager, Eastern Region
Brian Levine
71 W. 23rd St., New York, NY 10010
• phone (212)242-7447 • fax (212)242-7574
• email blevine@sigs.com

Advertising Sales Manager, Western Region
Jeffrey Wainhause
71 W. 23rd St., New York, NY 10010
• phone (212)242-7447 • fax (212)242-7574
• email jwainhause@sigs.com

Advertising Sales Representative for Development Solutions Buyer's Guide
Sarah Peyser
71 W. 23rd St., New York, NY 10010
• phone (212)242-7447 • fax (212)242-7574
• email speyser@sigs.com

ADMINISTRATION
Assistant to the President
Lynn Hazlewood

CIRCULATION
Circulation Director
Micki LaPorte

Assistant Circulation Manager
Adriel Marra

Circulation Assistant
Amit Bholasing

FINANCE
Controller
David Chatterpaul

Accounts Receivable Credit Manager
Rey Shields

Accounts Payable
Bibi Budhram

MANUFACTURING
Director of Production and Manufacturing
Elisa Varian

Assistant Production Manager
Michele Costabile

MARKETING
Director of Marketing Publications
Allyn Tracy Heck

Art Director, Marketing
Tom Jezek

Marketing Managers
Robin B. Schultz
Cheryl Cains

WORLDWIDE OFFICES
Jane Major, SIGS Ltd., Brocus House,
Parkgate Road, Newdigate, Surrey RH5 5AH, UK
• phone 011.44.1.306.631.331 • fax 011.44.1.306.631.696
• email enquiries@sigs.com

Jürgen Tillack, SIGS Conferences GmbH, Hauptstraße
293-297, D-51465 Bergisch Gladbach, Germany
• phone 49.2202.9372.0 • fax 49.2202.9372.2
• email 100634,2070@compuserve.com

buffers support numerous methods for inserting, appending, and deleting characters in the string, and can be converted easily to and from String objects.

The compiler uses the StringBuffer class internally to implement the String class' concatenation operator. Wherever an expression occurs using the string concatenation operator, the compiler replaces it with equivalent operations on a temporary StringBuffer object. For example, this line:

```
String s = "You have " + n + " new messages";
```

is converted by the compiler to the following statement:

```
String s = new StringBuffer("You have ").append(n).
    append(" new messages").toString();
```

A temporary StringBuffer object is first initialized with the leftmost operand. The remaining arguments are then appended to the string buffer using its append method.* The completed StringBuffer object is finally converted back to a String object using the toString method.

This translation of the original expression results in the construction of a temporary StringBuffer object. However, this is a cost that may be avoided if the arguments to the string concatenation operator are all constants. If the compiler supports it, operands based on constant values are concatenated at compile time and the construction of a StringBuffer object at runtime is circumvented. A compiler that supports the concatenation of strings based on constant expressions (see Table 1) would convert this expression, containing only constants:

```
"You have " + 5 + " new messages"
```

directly to this string literal:

```
"You have 5 new messages"
```

Because it requires the construction of a temporary string concatenation (when the compiler cannot eliminate it) incurs a fixed runtime cost per expression. If an expression containing multiple concatenation operators is split into independent statements, the cost of constructing a temporary is incurred by each statement instead of being incurred once and amortized across all concatenation operators in the expression. For example, the previous example split into three statements:

```
String s = "You have ";
s += 5;
s += " new messages";
```

prevents the compiler from concatenating the constants at compile-time and incurs the construction of two StringBuffer temporaries to boot!

The benefits of attempting to build composite strings using the fewest possible statements are greatest when the components are all constant values, as it gives the compiler the opportunity to replace the entire expression with a single string literal at compile-time.

Even when the string concatenation expression contains values computed at runtime, there is still considerable advantage in attempting to concatenate as many components together in the same expression. In this block of code, which displays the program's arguments, two of the components, i and args[i], are not known at compile time, so the concatenation would have to be deferred until the program executes even if the string had been built using a single statement. Nevertheless, splitting the concatenation into multiple statements still incurs an unnecessary penalty:

```
for (int i = 0; i < args.length; ++i) {
    String s = "args[";
    s += i;
    s += "] = \"";
    s += args[i];
    s += "\"";
    System.out.println(s);
}
```

Consider what happens to the body of the loop when the compiler re-expresses the string concatenation in terms of StringBuffer operations:

```
String s = "args[";
s = new StringBuffer(s).append(i).toString();
s = new StringBuffer(s).append("] = \"").toString();
s = new StringBuffer(s).append(args[i]).toString();
s = new StringBuffer(s).append("\"").toString();
```

This results in four StringBuffer and four String temporary objects being constructed! This is needlessly wasteful. The simple solution is to build the string with a single expression:

```
String s = "args[" + i + "] = \"" + args[i] + "\"";
```

This expression, once reworded by the compiler, is still considerably more compact than the previously translated code:

```
String s = new StringBuffer("args[").append(i).
    append("] = \"").append(args[i]).append("\"").
     toString();
```

There will be only two objects (one a temporary) created by this expression—a great improvement on the nine objects previously constructed. Depending on the JVM, the differ-

* Where one operand to the string concatenation operator is a String object and the other is not, *string conversion* is employed to convert the non-string argument to a string. Reference types are simply converted by invoking their toString methods. Conversion of primitive types necessitates the intermediate step of constructing a temporary instance of one of the *wrapper classes*—Boolean, Integer, Float, etc.—against which the toString method may be applied.

ence in performance between this formulation and the former one varies between a factor of 3 and a factor of 10.†

However, for a variety of reasons, it may not be possible to combine all the parts of a string together in one expression. In that case, imitate the behavior of the compiler by manually creating a StringBuffer object to hold the intermediate string values. Once the string is fully formed in the buffer, it may be converted to a String object with toString.

```
StringBuffer buf = new StringBuffer("n^19=");
double power = Math.pow(n, 19);
buf.append(power);
buf.append(" n^20=");
buf.append(power *= n);
String s = buf.toString();
```

This approach constructs only one temporary object—the string buffer—and so represents a significant time saving over multiple uses of the string concatenation += operator.

MAXIM 13: ALWAYS PARENTHESIZE SUB-EXPRESSIONS WHEN CONCATENATING WITH STRINGS

The compiler's somewhat mechanical conversion of string concatenation operators to StringBuffer operations can cause some surprises. For example, what output do you think the programmer intended this statement to produce?

```
System.out.println("When does 2 + 2 = " + 2 + 2);
```

Because string concatenation is used to "add" the numeric values, the statement does not produce the conventional answer of "4":

```
When does 2 + 2 = 22
```

The cause of this surprising output is that the + operator, representing both string concatenation and arithmetic addition associates left to right. So the expression:

```
w + x + y + z
```

is evaluated as

```
((w + x) + y) + z
```

If either w or x are strings, the other is converted to a string if required and the two strings are concatenated. The concatenation yields another string to which y and then z are further concatenated (after suitable conversions have been applied).

A similar problem arises when the numeric expression contains operators of lower precedence than the string concatenation operator:

```
str = "n >>> 4 = " + n >>> 4;    // compile error
```

In this instance, the right shift with zero-extension operator (>>>) has a lower precedence than the string concatenation operator. Evaluation of the first sub-expression (assuming n equal to 100) produces:

```
"n >>> 4 = 100" >>> 4
```

The >>> operator may not be validly applied to a String object, so a compile error results. This situation is less troublesome than the first, though, because it cannot go unnoticed.

As a matter of defensive programming, always parenthesize numeric and boolean expressions where they are to be concatenated with strings.

```
System.out.println("When does 2 + 2 = " + (2 + 2));
System.out.println("File is empty: " + (length == 0));
```

Even in cases where the arithmetic operators involved in a numeric expression all have precedence higher than the + operator, parenthesize the expression so that lines like this:

```
System.out.println("2n = " + n * 2);
```

do not change radically in meaning when replaced with arithmetically equivalent forms:

```
System.out.println("2n = " + n + n);    // intended?
System.out.println("2n = " + (n + n)); // intent clear
```

MAXIM 14: AVOID THE String.intern METHOD

There is one last thing to be said about improving the performance of string operations, and I know that if I don't say it, I will be deluged with email. So here goes: it regards the use of the String.intern method.

The intern method maintains a pool of unique String instances. When intern is invoked against a String object, the pool is searched (using equals) for an instance with an identical value. If one is found, the String object from the pool is returned. Otherwise, this String object is added to the pool and a reference to it is returned.

It follows that for any two strings s and t, s.intern() == t.intern() and s.equals(t) == true are equivalent tests (JLS 1.0, §20.12.47).

The application in Listing 1 reads a Java source file and counts how many of Java's 47 keywords occur within it. The keywords are stored in a string array and the tokens of the source file are compared sequentially against each of the array elements:

```
for (int t = 0; t < tokens.length; ++t)
    for (int k = 0; k < keywords.length; ++k)
        if (tokens[t].equals(keywords[k])) {
            ++counts[k];
            break;
        }
```

† Ironically, it is the immutable nature of String objects that allows the StringBuffer class to implement a very efficient toString method, which is a part of the reason why the difference is not much greater.

Introducing

XML WORLD

Increasing Productivity with XML

Conference

May 24-27 1999

Exhibition
May 25-26, 1999

Austin Marriott at the Capital
Austin, TX

3 Tracks, 30 Sessions, Hundreds of Answers

Introductory Background

XML is the technology of choice for exchanging structured information over the Internet, Intranets and Extranets. Web-based commerce, enterprise application integration and software system design are now set to take advantage of the many benefits of XML. This wide appeal will create standard infrastructure components, a large market for supporting tools and services, and widespread opportunity to define domain specific standards. **XML World** provides, for the first time, a forum for ideas and advances for Web application designers, IS professionals, and software engineers/developers.

Who should attend **XML World**

- Web page designers
- Ecommence service operations and developers
- Corporate Web masters
- ERP application managers and developers
- Supply-chain management systems designers
- Workflow application designers
- DBMS developers
- Internet application developers
- Distributed application developers

Technical Chair

Kevin Dick is a recognized industry expert in the use of object, Internet and XML technologies. Since founding his own consulting company (KDA) in 1994 he has helped over a dozen organizations.

Stay Current and Ahead of the Competition with these 3 Tracks

XML on the Web

Discover the benefits of XML over HTML for distributing and cataloging information. See how these benefits make XML technology the ideal format for on-line catalog, comparison shopping, and product configuration information. Enhance the capabilities of your Web applications with XML.

Topics Covered:

- Designing Web documents DTDs
- Using XML style sheets
- Using XML links
- XML browsers
- Searching XML documents
- XML e-commerce standards
- Managing documents, DTDs and style sheets

XML on the Enterprise

Learn how XML facilities data interchange, ERP application interchange, and supply-chain management. Because XML is a document-oriented language, using it as the lingua franca for business data enables workflow applications that cooperate smoothly in backend systems. This track delivers the answers you need to take advantage of XML's information integration and workflow potential.

Topics Covered

- Data interchange
- ERP application interchanges
- XML/EDI solutions
- Workflow
- Knowledge management
- Developing industry-standard DTDs
- Content management

Implementing XML Based Software

See why XML is the new requirement for many applications, including database management systems, ERP applications, and Ecommence systems. The availability of XML components and tools deliver development leverage that makes it a compelling technological choice. Learn why XML is cost-effective for implementing new Internet and distributed applications.

Topics Covered:

- XML specification
- XML parsers and validators
- Document object model (DOM)
- Designing XML based protocols
- Implementing XML based database interfaces
- Implementing XML based packaged application interfaces
- Implementing XML based distributed object systems

SIGS CONFERENCES

For information contact:

conference registration: Jonathan Goldberg, at (212)242-7447 ext. 123; or by email at jgoldberg@sigs.com.
exhibiting: Dennis Leavey, at (212)242-7447 ext. 136; or by email at dleavey@sigs.com

Listing 1.

Keyword counting application.

```
import java.io.*;
import java.util.*;

public class CountKeywords {
    public static void main(String[] args) {
        try {
            String[] tokens = tokenize(args[0]);
            int[] counts = countKeywords(tokens);

            for (int i = 0; i < keywords.length; ++i)
                if (counts[i] != 0)
                    System.out.println(keywords[i] + ": " + counts[i]);
        } catch (FileNotFoundException e){
            System.out.println("Cannot open file " + args[0]);
        }
    }

    static String[] tokenize(String file) throws FileNotFoundException {
        FileInputStream fin = new FileInputStream(file);
        DataInputStream in = new DataInputStream(fin);

        Vector tokens = new Vector();
        String line;

        for (;;) {
            try {
                if ((line = in.readLine()) == null)
                    break;
            } catch (IOException e) {
                break;
            }

            StringTokenizer st = new StringTokenizer(line, " \t\r\n");

            while (st.hasMoreElements())
                tokens.addElement(st.nextToken());
        }

        String[] s = new String[tokens.size()];
        tokens.copyInto(s);
        return s;
    }

    static int[] countKeywords(String[] tokens) {
        int[] counts = new int[keywords.length];

        for (int t = 0; t < tokens.length; ++t)
            for (int k = 0; k < keywords.length; ++k)
                if (tokens[t].equals(keywords[k])) {
                    ++counts[k];
                    break;
                }

        return counts;
    }

    static final String[] keywords = {
        "abstract", "boolean", "break", "byte", "case", "catch",
        "char", "class", "const", "continue", "default", "do",
        "double", "else", "extends", "final", "finally", "float",
        "for", "goto", "if", "implements", "import", "instanceof",
        "int", "interface", "long", "native", "new", "package",
        "private", "protected", "public", "return", "short",
        "static", "super", "switch", "synchronized", "this",
        "throw", "throws", "transient", "try", "void", "volatile",
        "while"
    };
}
```

In *The Java Programming Language, 2nd ed.* (Addison-Wesley, 1997), Ken Arnold and James Gosling describe a technique based on the intern method that can be used to improve the performance of string matching operations such as this one:

> However, any two strings with the same contents return the same String object from intern, which enables you to compare string *references* to test equality, instead of the slower test of string *contents*...Dealing with the results of intern makes comparing object references equivalent to comparing string contents, but much faster (§8.3, p. 166).

Applying that technique to the keyword counting application, the token string is initially *interned* so that the equals test may be replaced with a reference comparison between the interned token string and the current keyword string:

```
for (int t = 0; t < tokens.length; ++t) {
    tokens[t] = tokens[t].intern();

    for (int k = 0; k < keywords.length; ++k)
        if (tokens[t] == keywords[k]) {
            ++counts[k];
            break;
        }
}
```

Although the JLS (§20.12.47) states that, "All literal strings and string-valued constant expressions are interned", the program now fails to locate any keywords when running under JDK 1.0.2 (see Table 1) because the string literals in the keywords array have not been interned. A static initializer added after the declaration of the keywords array can be used to intern each of its string elements:

```
static {
    for (int i = 0; i < keywords.length; ++i)
        keywords[i] = keywords[i].intern();
}
```

Now the application works just the same as before. But by how much has this improved performance? Table 2 compares the times taken to count the keywords in the application's own source code using both approaches and a variety of runtime environments.

The results are quite surprising. The intern method approach fared well against plain string comparison with the older runtimes, but with more recent versions, it actually increased the execution time! Furthermore, when you consider the relative complexity of reference and string comparison,‡ why was the intern method, at its best, only improving on the equals method

by a factor of two?

The answer is that the intern method is a relatively expensive operation—on some JVMs, as costly as 48 string comparisons! This cost has to be recouped later by the savings made in the reference comparisons. (With the JVMs I tested, the break-even point varied between 9 and 52 comparisons.)

The intern method uses a hashtable internally to manage its unique references. For set membership operations on unbounded domains, hashtables are usually the highest performing data organizations. The performance of this application can be improved by making its use of a hashtable explicit.

First, a Hashtable variable is added to the CountKeywords class and then a static initializer is used to load the keywords and their index values:

```
static private Hashtable hashtable = new Hashtable();

static {
    for (int i = 0; i < keywords.length; ++i)
        hashtable.put(keywords[i], new Integer(i));
}
```

Any token may be identified as a keyword (in virtually constant time) by attempting to retrieve it from the hashtable:

```
for (int t = 0; t < tokens.length; ++t) {
    Integer k = (Integer) hashtable.get(tokens[t]);

    if (k != null)
        ++counts[k.intValue()];
}
```

Table 2 shows the performance improvement. In all cases, this out-performs both previous approaches! Why wasn't the intern method able to achieve the same high performance? First, once the intern method either located the token string in its hashtable or added it to it, the calling method still had to search its own keywords array with the interned sting. Second, the explicit Hashtable object only ever contains the keyword strings, whereas the intern method's hashtable has every unique token from the source file (whether keyword or not) added to it. It may take a considerable time to add these symbols to the hashtable.

The hashtable may also have difficulty managing so many strings. JDK 1.1.7, for example, crashes with this message if you attempt to intern more than 30,000 unique strings:

```
*** panic: 16-bit string hashtable overflow
abnormal program termination
```

‡ The relative execution times of reference and string comparisons depend on the host JVM. Reference comparison requires a mere three JVM instructions (two aloads and an if_acmpeq) so even in a best-case scenario (comparing two strings that differ on the first character) string comparison will be significantly more expensive on current JDKs, between 4 and 12 times.

Table 2. Keyword counting application benchmark figures.

	EQUALS	intern	Hashtable
JDK 1.0.2	56.6	26.7	7.5
JDK 1.1.1	25.7	8.3	4.0
JDK 1.1.5	31.5	8.7	3.3
JDK 1.1.6	1.5	2.2	0.7
JDK 1.1.7	1.7	2.1	0.8
JDK 1.2 beta 3	27.4	6.9	3.0
JDK 1.2 beta 4	1.4	0.9	0.5

The figures record the time taken, in milliseconds, by the string matching method of the keyword counting application (countKeywords) to count the keywords in the application's own source code using the three equivalent string matching algorithms.

Another reason you may wish to avoid the intern method is that, unlike the other two approaches, it cannot be adapted to perform case-insensitive searching.

The quickest way to search a collection of 10 or fewer strings is to place them in an array and iterate through them comparing with equals. For more strings than that, use a Hashtable object.

Regardless of the runtime environment, the intern method gives comparable or worse performance than equals when the number of strings is 10 or fewer and significantly worse performance than a hashtable when it is greater.

CONCLUSION

The decision by the Java language creators to implement strings as immutable objects was an inspired one, with numerous positive spin-offs. One such benefit was that it permitted string literals to be incarnated in Java as fully functioning, *bona fide* objects.

The tight integration of String objects into the language and the optimizations made possible by their immutable nature are benefits uniquely available to the Java programmer. Couple this with the techniques presented in this article and your strings will be truly supercharged.

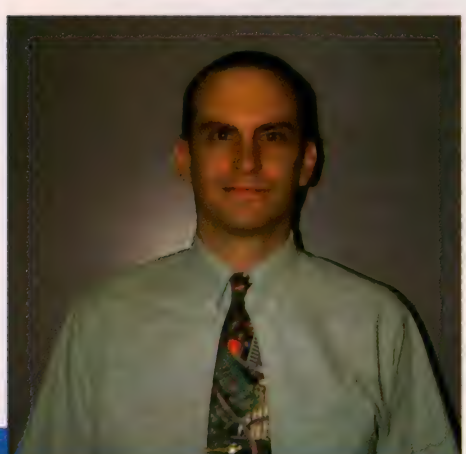
SOURCE CODE ONLINE

Source code for the benchmarking applications as well as complete implementations of the CountKeywords application using all three approaches is available on the Internet from the Effective Java page. ■

URL

Effective Java Web Page
http://effectivejava.com/column/superstrings

Client-Side Processing?



Steven W. Disbrow
diz@chattanooga.net

ONE OF THE main reasons JavaScript exists is to take some of the load off the server and put it on the client. For most folks, this means one thing: data validation. JavaScript is used to validate the data in an HTML form and that data is then sent to the server where some process uses it to generate a result page that is sent back to the client.

However, JavaScript can also be used to turn an HTML page into a simple client-side process. This is accomplished by sending information from one HTML file to another via a series of common gateway interface (CGI)-style parameters. You then use JavaScript to retrieve and process these parameters on the client.

THE SEARCH STRING

One style of CGI parameters you might be familiar with is a list of parameters at the end of a URL, like this:

```
http://www.asite.com/cgi-bin/query.app?thename=
steve&age=33
```

In this case, the first part of the URL (everything to the left of the question mark "?") is the URL of our server-side process. Everything to the right of the question mark is called the *search* or *query* string. (I'll call it the "search" string for the remainder of this article...because that's what it's called in JavaScript.) This is where we find our parameters arranged in "name=value" pairs. In this example, we have two parameters: "thename" (which has a value of "steve"), and "age" (which has a value of "33"). The ampersand ("&") separates the parameters from each other while the equal sign ("=") separates a parameter's name from its value.

Now, in this example, I've specified a fictional server-side process called "query.app" that's found in the "cgi-bin" directory of the Web server. However, I can easily change this URL to point to another HTML file, like so:

```
http://www.asite.com/aFile.htm?thename=steve&age=33
```

In this case, the Web server will load the file "aFile.htm" and the two parameters will be available for it to process. Of course, that processing will be done with JavaScript. (I should note here that some Web servers won't allow these sorts of shenanigans unless the target HTML file [the one you are passing the parameters to] has the "execute" permission associated with it. On the other hand, most Web servers don't care. After all, it's just an HTML file and any "executing" it does will be on the *client*, not the server.)

BUILDING THE SEARCH STRING

Before we look at the actual processing of our parameters, we need to look at how we actually create these search strings that we pass around.

The simplest way to create a search string is to create an HTML form that uses the "GET" method to load our HTML file. The following form will create the search string we just looked at when the user clicks the "Submit" button:

```
<form name="myform" method="GET"
action="http://www.asite.com/aFile.htm">
<input type="text" name="thename">
<input type="text" name="age">
<input type="submit" value="Submit">
</form>
```

Now, you might not have used the "GET" method very often. That's probably a good thing. The get method doesn't allow you to pass as much information as the "POST" method and, worst of all, every bit of information you pass via the "GET" method shows up in the resulting

Steven Disbrow is the owner of EGO Systems, a computer consulting firm in Chattanooga, TN. In the past, he has worked as a freelance programmer, Webmaster, editor, writer, and technical instructor.

URL. So, if you pass, say, a credit card number, it will show up in a *very* human-readable format for the whole world to see. (The moral is this: "Don't ever pass confidential or sensitive information this way!")

Even so, the "GET" method is what we need to use because the "POST" method simply doesn't make parameters available to client-side files (like HTML files).

BUILD IT YOURSELF

Another way to get a proper search string is to use JavaScript to build it yourself. After you have the search string built, you can tack it on to the end of a URL and assign the whole thing to the document.location.href property of the window or frame you want the HTML file loaded into.

(As you will recall, the "document" object is a JavaScript object that represents the HTML document that is currently loaded in the browser. The "location" object is similar to the location bar in a browser in that it holds the URL of the currently loaded page. The "href" property of the location object is a string representation of that URL. By assigning a new string value to the href property, you actually tell the browser to load a new page.)

For an example of this, consider Listing 1, the iForm.htm file. This file contains a simple form to collect the name and age of the user. In addition to the text boxes in this form, there are two buttons. The first is a submit button that uses the "GET" method to transmit the information to the iDisplay.htm file (see Listing 2). The second is a simple button that calls a JavaScript function, doBuildAndSubmit(), when it is clicked.

This function pulls the information out of our form and uses it to create a valid search string. (Note that an ampersand is placed between each parameter in the search string.) After the search string is created, we append it to the end of our URL and assign the whole thing to the document.location.href property. (To make the new file show up in a different frame, put "parent.framename." in front of "document" in the assignment statement. If you want the new file to show up in a different frame or window when using the "GET" method, just specify a "target" attribute in your form. Listing 1 has an example of this.)

A HEX ON YE!

One other thing to notice about this function is that it makes use of a function called esc(). This function's job is to take any special characters in our parameters and change them into their hexadecimal representations. These hexadecimal codes appear as a percent sign followed by two hexadecimal digits in the resulting search string. This is necessary because some characters (most notably spaces) can cause information to be lost when they pass through the common gateway interface. So, we "escape" these values to make sure no information is lost.

When you use the "GET" method, the browser escapes all the values for you. But when you build your own search string, you have to do this task yourself. To accomplish this, JavaScript provides the escape() method. You pass it a string and it returns a new string with almost all of the special characters converted into their hexadecimal equivalents.

So, why do we need the esc() function? It's because at least one special character isn't properly converted into hexadecimal format by the escape() method. This character is the "+" char-

Listing 1.

iForm.htm.

```
<html>
<head>
<script language=javascript>
// This function enhances the built-in escape method by
// additionally escaping "+" characters.
function esc( theString) {
    result = escape( theString)
    i = result.indexOf( "+")
    while (i != -1) {
        // "%2B" is the escaped version of "+"
        result = result.substring(0, i) + "%2B" +
        result.substring(i + 1)
        i = result.indexOf( "+", i + 1)
        }
    return result
    }

//Build a search string and use it to invoke a HTML file
function doBuildAndSubmit(f) {
    // This is where you would check the info for validity
    // If the info is valid, continue on with the task at hand...

    // Pull the values out of the text boxes
    theName = f.thename.value;
    theAge = f.age.value;

    // Build the search string that we'll pass to the next file
    sString = "thename=" + esc( theName);
    sString += "&age=" + esc( theAge);

    // Pass the info to the next file
    document.location.href = "iDisplay.htm?" + sString
    }
</script>
</head>
<body bgcolor="white">
<form name="info" method="get" target="newWindow" action="idisplay.htm">
Enter your name: <input type="text" name="thename"><br>
Enter your age: <input type="text" name="age"><br>
<input type="submit" value="Submit">
<input type="button" value="Build & Submit"
onClick="doBuildAndSubmit(this.form)">
</form>
</body>
</html>
```

Listing 2.

iDisplay.htm.

```
<html>
<head>
<script language=javascript>
// This function returns the value associated with a named parameter
// in the search portion of a URL. Note that this function
// assumes that this part of the URL has been passed through the
// esc() function that is defined in the iForm.htm file.
function getSearchParam( pName) {
    // Get the search string that was passed to this document
    // Note that we peel off the first character, which is a "?"
    var remainder = document.location.search.substring(1)

    // if the search string was built by the browser (via "submit"),
    // spaces will be replaced by "+" characters. So, we change these
    // back to spaces.
    i = remainder.indexOf( "+")
    while (i != -1) {
        remainder = remainder.substring(0, i) + " " +
        remainder.substring(i + 1)
        i = remainder.indexOf( "+", i + 1)
        }

    result = "";
    while (remainder != "") {
        // Find the item separator (&) if it exists
        itemEnd = remainder.indexOf( '&');
        if (itemEnd == -1)
            itemEnd = remainder.length;

        // extract this item
        thisItem = remainder.substring( 0, itemEnd);

        // get the name of this item
        nameEnd = thisItem.indexOf( '=');
        thisName = thisItem.substring( 0, nameEnd);

        // Is it what we wanted?
        if (thisName == pName) {
            // yes! Get and return the value
            result = thisItem.substring( nameEnd + 1);
            break;
            }
        else {
            // Break off the rest of the
            // search string and do it again
            remainder = remainder.substring( itemEnd + 1,
            remainder.length);
            }
        }
    return unescape( result);
    }
// Remove the comment from the
// next line to see the contents of the href
// alert (document.location.href)
</script>
</head>
<body bgcolor="white">
<H3>The name is
<script language="javascript">
document.write( "" + getSearchParam( "thename") + "");
</script>
</H3>
<H3>The age is
<script language="javascript">
document.write( "" + getSearchParam( "age") + "");
</script>
</H3>
</body>
</html>
```

acter. The problem is that instead of changing spaces into hexadecimal format, the "GET" method replaces them with "+"s. (Actual "+" characters are converted to hexadecimal format.) On the other hand, the escape() method *does* convert spaces to hexadecimal format, but it leaves "+" characters untouched. (Note that the "&" and "=" characters *would* be converted to hexadecimal. Because we need those to separate our parameters, we don't pass those to the esc() function or the escape() method.)

In either case, the "+" characters travel through the common gateway interface with no problem. But, if you are trying to write a client-side page that will receive this info from either a "GET" method call or a hand-built search string, you don't know if that "+" you received started out as a space or as an actual "+".

The solution to this problem comes in two pieces. When creating your search string you use a wrapper function (the esc() function in this case) that uses the escape() method to convert any spaces (or other special characters) to their hexadecimal values. You then go through this resulting "escaped" string and manually convert any "+" characters to their hexadecimal value (which is "%2B"). The resulting value can then be passed to the HTML file on the other end where it can be un-escaped (using the unescape() method) to recreate the original value.

The second half of the solution is in that HTML file on the other end. In Listing 2, take a look at the getSearchParam() function. This function takes the name of a search string parameter and returns the value that was associated with it.

The first thing it does is to retrieve the contents of the document.location.search property. As you might guess, this property contains the search string that was passed by the caller. Unfortunately, it also contains the "?" at the front, so we have to strip that off with a call to the substring() method. (Note that all of this information is also in the document.location.href property. But, this property also contains everything to the *left* of the "?", and we don't need any of that.)

Next, we move through the search string and replace any "+" characters with spaces. We can do this with confidence because the only way a "+" character will appear in the search string is if the search string was built by the browser (via the "GET" method). (Remember, if *we* built the search string all the "+" characters were converted to hexadecimal values, and they will still be hexadecimal values at this point.)

After that, it's a simple matter of string parsing to locate the requested parameter name and it's associated value. Once the value is found, we unescape() it and return it to the caller.

To test all this out, first load iForm.htm into your Web browser and enter your name and age. Then click on either the "Submit" or "Build & Submit" buttons. The information you entered should then appear on the next HTML page you see.

(If you are using Internet Explorer to test the files shown in Listings 1 and 2, be sure you load them using the HTTP protocol. (In other words, load them via a Web server.) If you simply load them as files off a disk, Internet Explorer won't pass the parameters correctly. Netscape Navigator doesn't care one way or the other.)

BUT WHY?

Aside from the fact that you are moving even more work off of the server, why bother with all of this HTML-file to HTML-file nonsense? Why not just send the data to the server and let the Java programmers worry about it?

Honestly, what we've looked at here isn't always a good solution. You certainly can't use this technique to plug data into a server-side database. But, you *can* use it to feed parameters to a Java applet that you want to appear in another browser window.

Another good use of this technique comes up in the venerable shopping cart scenario. When the user clicks the order button, you can send everything from their shopping cart to another HTML file where it's echoed back in a more traditional format (perhaps as an invoice). They can look this over and make any needed changes. This page could also include a "Submit" button that actually initiates the secure transaction with the server.

Knowing how to build (and tear apart) your own search strings with JavaScript has other advantages as well. For example, you can use this technique to place additional parameters (perhaps one or more flag values) in a search string, without having to code corresponding HTML form elements. These additional parameters will then be passed to the target process just as if they had appeared in the form.

TRY IT!

With a little thought, you should be able to find quite a few uses for this technique on your own Web site. If you can lighten the load for even one over-worked server-side process, it will be worth the effort! ■

Find all *Java Report's* listings online at www.javareport.com

OO SOFTWARE *continued from page 61*

easily applied to a particular software design situation. A lot of people have been developing software for some time now. Patterns help developers learn and use the "timeless" designs of the experts. Without patterns, developers reinvent designs to solve recurring problems. Often, those designs will not have the experience necessary to be optimal. Designs also take time to devise from scratch, where an optimal solution may already be documented and time-tested in a pattern. Finally, patterns help convey a large amount of information in a very short time. Each pattern has a short but distinguished name, so those developers versed in patterns can rapidly discuss design alternatives amongst themselves.

CONCLUSION

The Java language, its APIs, and recent Java development environments present a powerful toolset for software development. Unfortunately, this toolset can very quickly cause the creation of a lot of bad code if not held in check by good OO software development technique. I've described the most widely used techniques in OO software development today. These techniques should ensure that whatever code you produce will first and foremost satisfy user requirements and second be designed to stand the test of time.

REFERENCES

1. Jacobson, I. *et al. Object-Oriented Software Engineering—A Use Case Driven Approach*, ACM Press/Addison-Wesley, 1992.
2. Bellin, D. and S. Suchman Simone, *The CRC Card Book* (Object Technology Series), Addison-Wesley, 1997.
3. Wilkinson, N. M. *Using CRC Cards: An Informal Approach to Object-Oriented Development (Advances in Object Technology*, No. 6), SIGS Books & Multimedia, 1995.
4. Wirfs-Brock, R. *et al. Designing Object-Oriented Software*, Prentice Hall, 1990.
5. Fowler, M. and K. Scott, *UML Distilled: Applying the Standard Modeling Language*, Addison-Wesley, 1997.
6. Fowler, M. *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997.
7. Gamma, E. *et al. Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994. ■

Todd Lauinger is an I/T Consultant at Cargill Inc., Minneapolis, MN. He can be contacted at Todd_Lauinger@bigfoot.com.

Designing durable APIs in Java

DESIGNING FOR larger-scale systems can be quite a bit different than designing for small programs. In particular, the cost of making incompatible changes becomes very high. This is the case even within the same company—and when your product is library code available to clients, incompatible changes can be either difficult or impossible. This means that you have to design your code to be *durable*—where the APIs can withstand the test of time, and can be extended without breaking client code.

Mark Davis
mark@macchiato.com

Durability is not typically highlighted in object-oriented (OO) design books. In this new *Java Report* column we'll focus on particular techniques for ensuring that your Java APIs are durable. On the other side of the coin, there are a lot of non-durable APIs out in the world, so we'll also discuss *defensive* techniques for subclassers.

This column is the product of many years of wrestling with APIs for large OO software systems, first in C++ and then in Java. While my primary area of expertise in software is internationalization, I've managed architecture design and review teams for very high-level software on down to very low level. During this time, I've had the benefit of working with some really outstanding people, and will draw on what we learned in working through the issues presented here.

Dr. Mark Davis is a program director at IBM's Java Technology Center, Silicon Valley, CA, co-founder of the Unicode effort, and is the president of the Unicode Consortium.

RELIABLE SUPERCLASSES

Let's talk about classes for subclassing. Designing classes for subclassing is easy. Designing classes for *reliable* subclassing is not. It is very easy to muck this up: you can give your subclassers too much control, or too little control, or you can force subclasses to be *fragile*.

What do we mean by a fragile subclass? The specification for Java classes is customarily established by the javadoc documentation, which includes a text description and the public and protected method and field signatures. Any implementation change that stays within the spec is compatible. When you are subclassing a class, you may figure out a particular way to subclass based on fishing through the Java implementation to figure it out. But your subclass is then *fragile*—compatible changes can blow you out of the water.

Let's look at some of the problems and how to avoid them.

TOO MUCH CONTROL

So what is the problem with giving subclasses too much control? The disadvantage is that it ties your hands for future subclassing. For example, if you make your fields public, then you have no control over when they get changed and you can't decide that you want a different way of organizing your data later on. If the fields are dependent on one another, then you can't make sure that they are set in tandem or that they don't end up with inconsistent values.

PRIVACY ISSUES

Generally, it's a bad idea to have fields be anything but private. If they are private, you have complete control over access, and don't have to worry about users, subclassers, or other classes in your package doing unpleasant things behind your back. You can supply access with getters and setters having the appropriate access control: public, protected, private, or package-private.

Of course, there are many circumstances where it's reasonable to make the fields public. Look at **Point**, for example. It's simply a

collection of two simple fields x and y, which are directly accessible for speed. The best case for making fields public is when: (1) your class is final, (2) the fields are independent: one doesn't depend on another, and (3) access speed is crucial. Yet even here, a modern JIT will make a final getter or setter be just as fast as a public field, so there is often little reason to have public fields.

Don't make internal fields package-private. Package-private fields can be accessed and changed by any class in the package, intentionally or unintentionally, behind your back and a subclasser's back. A future implementation change in any other class in that package—with no API or implementation change to your class—could cause subclassers to malfunction. If package-private access is required, funnel it through a package-private getter or setter.

There are times when you want to give access to both subclassers and other classes in your package. Do this with two different methods, funneling through the protected method.

```
protected void setField(int value) {
 ...
}

void setFieldInternal(int value) { // package private
setField(value);
}
```

NOT ENOUGH CONTROL

Subclassers have insufficient control when they need to access or change private fields but can't. They also can have insufficient control when your class (or classes in your package) go through back doors. Let's take BorderLayout, for example.

Two of BorderLayout's fields are fine and accessible through public getters/setters: get/setHgap, get/setVgap. But it has five package-private fields, corresponding to the "North", "South", "East", "West", and "Center" parameters passed in to **Component**.add. Because these are inaccessible, the only way to subclass them is to have your own copies and override every **BorderLayout** method that either uses or sets them. This means duplicating the vast majority of code in the class. (Moreover, the fields are package-private, not private, which makes them more fragile.)

To avoid this problem in your development process, start by ensuring the class is final. As a part of your testing, construct a substantial subclass—one that significantly modifies the behavior of your class. Once a class is tested for subclassability, it can be modified to allow sufficient access to its data and final could be lifted. Remember, removing final is a compatible change—adding it is not!

Call yourself

A class can also not permit sufficient control when it does not call its own setter when it is changing its internal state. By doing that, subclassers are not able to reliably override the setter, other than overriding all the methods that change the state. Look at the following example:

```
public class AClass {
private int x;
public void someMethod() {
  ...
  x = 5 * x;                          (1)
  // setX(5 * getX());                (2)
  ...
}
public void setX(int newValue) {
  x = newValue;
}
public int getX() {
  return x;
 }
}
```

If you wrote line (1), your subclasser couldn't just override setX—he/she would have to override someMethod and duplicate all of its implementation. By writing line (2) instead, your subclasser only needs to override setX. Unless a setter or getter is final, and thus can't be subclassed, always call it instead of accessing the field directly.

Call your super

When you override methods, make sure that you call your superclass's method at some point. Unless that superclass method is clearly spec'ed out to not need overriding, a compatible change could break your subclass. Look at the following simple cases.

```
public class Foo {
    ....
    public void setName(String name) {
        this.name = name;
        hashCache = name.hashCode();          (1)
        }
        public int hashCode() {
            return hashCache;
    }
        protected String name;
        protected int hashCache;              (2)
}

class Foo2 extends Foo {
    ....
    public void setName(String name) {
        this.name = name;                     (3)
            // super.setName(name);           (4)
            containsSlash = name.indexOf('/') > 0;
```

```
    }
    boolean containsSlash;
}
```

In version 1, Foo does not have lines (1) and (2). If Foo2 uses line (3) then everything seems to work fine at first. But then along comes the author of Foo and makes a performance enhancement—all within spec—by adding (1) and (2). Foo2 then breaks, because its hashCode method now fails. If Foo2 is coded with line (4) instead, all is well in either case. (Of course, this also illustrates the value of having private classes—if name were private this wouldn't happen!)

DOCUMENTATION!

Except for trivial classes, the methods' signatures alone are not sufficient to tell someone how to subclass. If a class is designed to be subclassable, it must *document* how to subclass. With more complicated arrangements of objects, your objects' methods can be called by many other classes under a wide variety of circumstances. If you don't document how to subclass, it is very easy for subclassers to produce fragile subclasses. The two main problem areas are where:

- Overriding a method has no results, because it is not called internally.
- Overriding a method has unexpected results, because other coordinated methods need overriding at the same time.

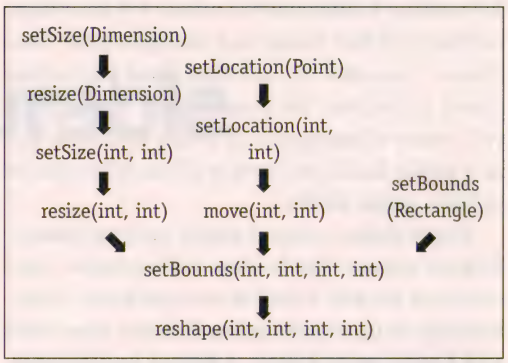
Which method to override?

Suppose I want to be notified whenever my **Component** changes its size. If **Component** were badly designed, it might change its size without calling an overridable method. Then I would be completely stuck.

Luckily, the designers of **Component** avoided that, and never set the size without calling an overridable method. But which ones? If we look at the signature for **Component**, we find a whole host of methods having to do with size and positioning.

- reshape(int, int, int, int) [Deprecated]
- resize(**Dimension**) [Deprecated]
- resize(int, int) [Deprecated]
- move(int, int)[Deprecated]
- setSize(int, int)
- setSize(**Dimension**)
- setBounds(int, int, int, int)
- setBounds(Rectangle)
- setLocation(int, int)
- setLocation(Point)

If you examine the Java 1.1.4 code, you will find the following call chains:



This is all of these methods funneled through one method: reshape(int, int, int, int). This is the primary method—all others are secondary methods and eventually just call the primary one. (Not always along an efficient path!) Because this method was deprecated as part of a name change, you will also find by examining the code that all* of the 1.1 Java code funnels through the new name, setBounds(int, int, int, int). This is excellent—we only have to override one method, *not* all of them.

However, this funneling behavior is not part of the specification. This means that it is fragile: Sun or any Java licensee could change this behavior and break your subclass. In this particular case, it is unlikely—but if you were paranoid you would not depend on unspecified behavior and override each of the methods.

Let's pull back a bit and see what you, as the superclass writer, can do to avoid this problem. In general, there are two ways to provide for reliable subclassing in this kind of case.

- Mark all secondary methods as final.
- Document all secondary methods with something like:

  "Do not override this method. To achieve the same effect, override XXX(...)."

In new code, do both of these. Marking a method final will let the compiler tell your subclassers they are not doing the right thing. Unfortunately, that also means that you can never fix an old method by marking it final without breaking compatibility. So for old methods that should not be overridden, the documentation alone will have to do.

Coordinated methods

Many methods are coordinated with one another. To effectively subclass, you need to override those methods as a group. For exam-

*Be careful to catch all the uses of the method in all of your classes. For example, there is a flaw in the JavaSoft renaming of reshape. **ScrollPane**.layout() calls reshape directly, and will thus not go through an overridden setBounds(). This will cause subclasses to malfunction unless they override both method names.

ple, let's suppose that you are building a component that needs to rearrange its contents in response to a change in size. You could start by overriding setBounds(), but you will find out that you shouldn't re-layout your component in response to this, you should override layout() and do the re-layout there. You will then find out that layout() is only called on subclasses of **Container**, not on arbitrary subclasses of **Component** (even though it would be faster to not have to test for subclasses).

So even though **Container** has more fields and API than you really wanted, you will have to extend it instead of **Component**. Of course, none of this is documented in the API, so it is a matter of fishing through the implementation to figure it out. You have no choice but to have a fragile subclass.

There is no language mechanism in Java to indicate that two method are coordinated. The only way to prevent this is to get it documented in the superclass API documentation.

SUMMARY

Here is a summary of the main guidelines for helping to make sure that your classes are designed for reliable subclassing.

- Keep your fields private (and not just package-private).
- Call your own getters/setters.
- Mark classes final, until you have confirmed that they are reliably subclassable.
- Mark methods final until you have confirmed that they are reliably overridable.
- Never trust that your class is reliably subclassable until you have carefully reviewed the design, and written and tested a significant subclass for it.
- Always document clearly how to subclass.

IN THE FUTURE

I welcome any feedback from readers, both on published columns and on topics they would like to see covered. Here are some of the topics I'm planning to cover with this column.

- *Using Abstract Classes and Interfaces*—When should you use an abstract class and when should you use an interface? The durability of your APIs can be compromised if you use interfaces where you should use abstract classes, and when you don't properly supply abstract superclasses.
- *Avoiding Object Creation*—Many articles have been written about how much excessive object creation can affect your program performance. The lifetime cost of an allocated object includes a number of costs you don't often consider. By avoiding "baking" object creation into your APIs, you can make them more durable and allow your clients to have better performance.
- *Defining Immutable Objects*—Immutable objects are a common pattern in the Java class libraries. Used properly, they can be very effective. However, it is very easy to overuse them or to define them incorrectly, leading to bugs that are quite difficult to track down.
- *Using Factory Methods*—Factory methods provide a useful alternative to constructors. They allow you much more control over the construction of instances of your class, giving you the opportunity to have faster and more durable construction of new objects.
- *Changing Method Names*—Suppose you find that you need to rename a method. It is surprisingly easy to make mistakes in how you do this; mistakes that will break either old or new subclasses. The correct methods for doing this will minimize the problems for your clients.
- *Reliable Serialization*—How do you modify a class, and still make sure that your old code can read your new serialized instances and your new code can still read your old serialized instances? The Java serialization mechanism handles a number of these issues for you, but there are some gotchas!

I try to draw as many examples as possible from the Java API (both 1.1 and 2.0). This is because those APIs are more familiar to people, *not* because they are bad. While there *are* flaws in the Java APIs—that is to be expected from any product of such complexity—you'd rather hit your window of opportunity than ship absolutely perfect APIs! ■

TEXT SEARCHING *continued from page 58*

code collation, see www.unicode.org/unicode/reports/tr10. ■

Laura Werner is a Senior Software Engineer and Project Leader for the Java Internationalization effort at the IBM Center for Java Technology in Cupertino, CA. She can be contacted at lwerner@us.ibm.com.

URLs

Java Developer Connection
developer.javasoft.com

IBM alphaWorks
www.alphaworks.ibm.com

Unicode Consortium
www.unicode.org

SCOTT'S SOLUTIONS *cont'd. from page 80*

unused to printing conventions, the 72 in the aforementioned calculations is the number of points per inch; essentially, we can treat all measurements in inches and multiply them by 72 to get the units we need.]

Similarly, we set the page format to landscape with the given paper, and now we pass the page format when we construct the printer job.

When the print dialog is presented to the user, the user may have the option of changing those options (depending on the platform-specific dialog); keep in mind that these options are default options only.

Now all that is left to do is to implement the print() method of our Printable class. In doing so, we must keep two points in mind:

- The print() method is responsible for returning a value that says whether or not a particular page exists. In the case of printing a file, that means that the print() method essentially must indicate when all the lines of the file have been printed.
- The print() method may be called multiple times to print the same page. However, the print() method will never be asked to print pages out of order: it will print page 0 one or more times, then page 1 one or more times, and so on.

Listing 2 shows a basic implementation of our print() method.

This is a somewhat large method for an example, but the basic flow is this:

- When we initialize the first page, it's the first time that we have a Graphics object, so we can go ahead and initialize the FontMetrics.
- If we just did this page, we must rewind the file to the last saved position so that we write out the same lines again.
- We translate the Graphics object to make our drawing easier. The Graphics object will include the page margins, and we don't want our code to worry about those. So instead, we translate by the starting X and Y co-ordinates, so that our drawing can be based on the familiar 0,0 coordinate system.
- Then we simply draw each line, incrementing our Y position each time through the loop. When we've either drawn the last line that will fit on the page or have reached the end of the file, we return PAGE_EXISTS to indicate that we successfully drew the page (that constant is part of the Printable interface).
- When we're asked to draw a page that is past the end of the file, we return NO_SUCH_PAGE to indicate that the requested page should not be printed.

And that, essentially, all there is to it. The resulting output will be somewhat plain, but it will contain the entire file. One way that we might make the output look a little better is to put a header on each page. Let's say that we insert a call to a printHeader() method after we set the color in the Graphics object and set the starting value of Y to include the space drawn by the header. A simple printHeader() method might look like this:

```
void printHeader(Graphics g,
                 PageFormat pf, int page) {
    int width =
    (int) pf.getImageableWidth();
    g.drawLine(0, lineHeight + 5,
        width, lineHeight + 5);
    g.drawString(
            name, 0, lineHeight);
    String s = "" + (page + 1);
    int len = fm.stringWidth(s);
    g.drawString(s,
            width - len, lineHeight);
}
```

This draws a simple line on top of the page; above the line we print the current file name and the current page number. Note that the printing system starts pages numbered at 0, so we add 1 to the page number before printing it out; someday, humans and computers might think the same way about such matters.

Similarly, we might add methods to draw footers or line numbers or anything else that we might desire. One final caveat: depending on

Listing 1.

Printing a file listing.

```
public class FilePrinter
             implements Printable {
    RandomAccessFile raf;
    Font f;
    FontMetrics fm;
    int lineHeight;
    String name;
    int lastPage;
    long lastPos;

    public FilePrinter(String name)
                throws IOException {
        raf = new RandomAccessFile(
                    name, "r");
        this.name = name;
        lastPage = -1;
    }

    public int print(Graphics g,
            PageFormat pf, int page) {
        ...
    }
}
```

the graphics system involved, the drawString() method isn't smart about tabs, so that you might want to use the method shown in Listing 3 to convert a string before printing it out.

The convertString() method might similarly wrap long lines or perform other useful substitutions.

This technique of printing is useful anytime that you need to print out something that takes more than one page. If you want to print a JTable, for instance, then you follow the basic premise outlined here: you create a subclass of JTable that implements the Printable interface, and in that print() method you only print as many rows as will fit on the page. The code to do that is only complicated in the sense that the JTable API is complicated; from a printing standpoint, it's exactly the same as what we've just shown.

One final word about printing, especially in Java 2, and that has to do with security. As most of us are aware, you cannot instantiate a PrintJob object in an applet due to Java's security model; the same is true for PrinterJob objects. In 1.1, you can work around that by signing the applet, assuming that your browser supports signed applets (and allows signed applets to print).

The Java 2 security model is more flexible in this regard. For the most part, we're interested in how this works presently with the Java Plug-in, because that's the only 1.2-based browser we have. To print with the plug-in, you must ask the user of the plug-in to enable your applet to print. This means that the user (or system administrator) must put the following entry into the java.policy file:

```
grant
codeBase "http://yourhost/" {
    permission java.lang.RuntimePermission
      "queuePrintJob";
};
```

You'll need to substitute the correct URL for the code base, and you may optionally add a signedBy attribute if you want to sign the applet. Note that the signedBy attribute is optional. However, using the 2.0-based security model, you can print from an unsigned applet as well as from a signed applet (assuming, of course, that the user or system administrator of the machine hosting the applet decides to allow such access).

Whether its from an applet or application, the 2.0-based printing system offers a powerful and flexible way to print reports, large tables, or any other page-based output. In fact, we've really only scratched the surface of this system: the API allows us to set different page formats and properties for different pages as well as managing a collection of pages. With Java 2, printing in Java has finally arrived. ■

Listing 2.

A basic implementation of the print() method.

```
public int print(Graphics g,
        PageFormat pf, int page) {
    try {
        if (page == 0) {
            f = new Font("Monospaced",
                    Font.PLAIN, 12);
            fm = g.getFontMetrics(f);
            lineHeight =
                        fm.getAscent();
        }
        if (lastPage == page)
            raf.seek(lastPos);
        else lastPos =
                raf.getFilePointer();
        if (raf.getFilePointer() ==
                    raf.length())
            return NO_SUCH_PAGE;
        g.setFont(f);
        g.translate(
            (int) pf.getImageableX(),
            (int) pf.getImageableY());
        g.setColor(Color.black);
        String s;
        int y = lineHeight + 10;
        int totalHeight =
        (int) pf.getImageableHeight();
        while (
        (s = raf.readLine()) != null
                && y < totalHeight) {
            if (s.length() != 0)
                g.drawString(s, 0, y);
            y += lineHeight;
        }
        lastPage = page;
        return PAGE_EXISTS;
    }
    catch (IOException ioe) {
        lastPage = -1;
        return NO_SUCH_PAGE;
    }
}
```

Listing 3.

Because the drawString() method isn't smart about tabs, you might want to use the following method to convert a string.

```
String convertString(String s) {
    StringBuffer sb =
            new StringBuffer();
    char c[] = s.toCharArray();
    int curChar = 1;
    for (int i = 0;
                i < c.length; i++) {
        if (c[i] == '\t') {
            while ((curChar % 8)!=0) {
                sb.append(" ");
                curChar++;
            }
        } else {
            sb.append(c[i]);
            curChar++;
        }
    }
    return new String(sb);
}
```

Printing in Java

This column is designed to give readers solutions to questions with a "how-to" bent. If you'd like to submit a question, send it to ***soaks@sigs.com*** *or visit* ***http://www.sigs.com/java/ask.html****. In each issue, we'll answer the most common questions about Java programming.*

Scott Oaks
soaks@sigs.com

IN THIS INSTALLMENT, we explore the issues surrounding printing in Java. We answer a variety of related questions, from the basics of how to print something in Java to more complex issues of printing things that span pages. Finally, we also touch on security issues related to printing.

As we answer these questions, we'll deal with the new printing API that is available in Java 2. If you're still using 1.1, then we'll point out a few of the differences as we go along, but to have the type of control that we really want over our output, we really need to utilize the new features of Java 2.

To begin, we need to decide what it is that we want to print. As an example, we'll print out a file listing by using the class in Listing 1.

The key here is that to print something, it must implement the Printable interface, which means it must provide a print method with the given signature. This is our first change from 1.1, when no interfaces were required. However, in 1.1 all AWT components have a print method with a different signature—that method simply requires a graphics object. The print method of the Component class still exists, but it uses the old 1.1-based PrintJob class rather than the newer, more flexible, PrinterJob class that we're about to look at.

Next, we need access to a printer job. We can then tell the printer job what it is supposed to print, and ask it to print it like this:

```
PrinterJob pj =
        PrinterJob.getPrinterJob();
pj.setPrintable(
            new FilePrinter("myfile"));
if (pj.printDialog())
    pj.print();
```

The second line of this code is how the printer job knows what to print; we must pass it an argument that is an instance of type Printable. Note that this procedure is very similar to the way in which we used to print in 1.1: in 1.1 we'd call the getPrintJob() method to return a PrintJob instance, and then we would call the print method on a particular object that we wanted to print.

However, the printer job class is much more powerful than the print job class. As you can guess from the interface of the print() method, the PrinterJob interface keeps track of which page you're on; with a PrintJob, you must keep track of paging manually.

In addition, the PrinterJob class allows us to set default page sizes and orientation. If you're printing a table, for example, you might set the default page orientation to landscape on legal-size paper:

```
PrinterJob pj;
Paper p;
PageFormat pf;

p = new Paper();
p.setSize(72 * 8.5, 72 * 14.);
p.setImageableArea(.5 * 72,
        .5 * 72, 72 * 7.5, 72 * 13);
pf = new PageFormat();
pf.setPaper(p);
pf.setOrientation(pf.LANDSCAPE);
pj = PrinterJob.getPrinterJob();
pj.setPrintable(
    new FilePrinter("myfile"), pf);
```

There's a lot more to the Paper and PageFormat classes that we won't cover here, but this shows the basic idea: on a piece of paper, you can set the imageable area to whatever size you want. In this case, we want 8.5- by 14-inch legal paper with a half-inch margin all around, so we set the size and imageable area of the paper accordingly. [If you're

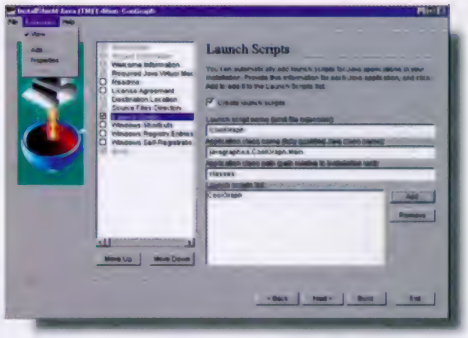
continued on page 78

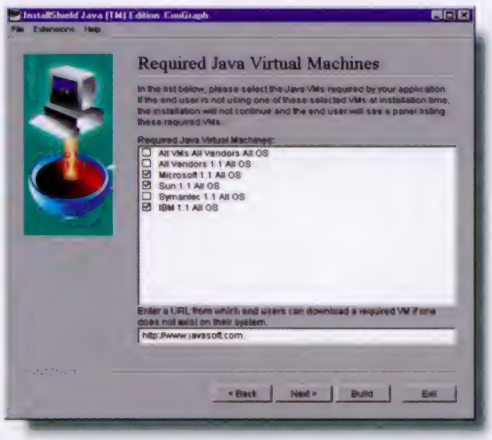
Scott Oaks is a Systems Engineer for Sun Microsystems, where he focuses on practical applications of Java technology. He is the co-author, with Henry Wong, of *Java Threads*.

InstallShield® JavaEdition™ Version 2

Extend the possibilities of your Java installations.

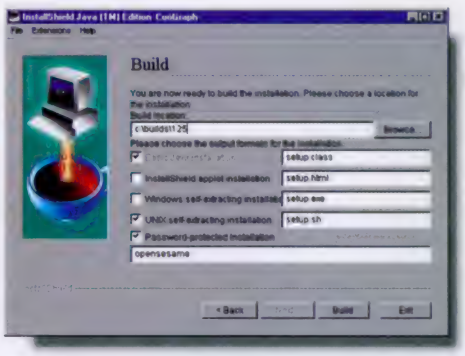
InstallShield Java Edition puts the power of Java in your hands to customize your installation. Using InstallShield Extensions for the Java platform, you can plug in your own Java classes or take advantage of pre-built examples. The integrated Extensions view lets you visually manage and rearrange the order of your installation process.





True cross-platform installations ... once and for all.

The single self-extracting CLASS file created by InstallShield Java Edition is readable by all operating systems with supported Java Virtual Machines (VM), version 1.1 or higher. The CLASS file can even be placed on the Web for easy download. The end-user experience is consistent in look, feel and functionality across all platforms.



Quickly create installations for easy distribution.

The InstallShield Java Edition builder walks you through every step toward creating a single CLASS file containing your installation and application. InstallShield Installation Applet lets your customers install the appropriate Java VM and your application directly from a Web page. You can also create Windows® and Unix self-extracting installations.

InstallShield® Software Corporation

The total solution for Java™ deployment.

Java developers have always been required to write and maintain individual installations for each of their target platforms. When it came to deployment, "write once, run anywhere" didn't always apply. Until now.

InstallShield® Java Edition provides developers with a visual tool designed to create a single Java installation that's consistent in look, feel and functionality across multiple platforms. New InstallShield Extensions for the Java platform bring the power of Java to installation building, allowing you to plug your own Java-based extensions into your installation. And the InstallShield Installation Applet lets you create a seamless cross-platform launch site for your installation. Best of all, InstallShield Java Edition is authored by InstallShield Software Corporation, the world leader in installation and distribution technology.

Exclusive features in InstallShield Java Edition 2

- True cross-platform installations — no need for multiple installations for individual platforms
- InstallShield Extensions for the Java platform give you the power of Java to build your installations — plug in your own Java-based extensions or use one of the pre-built examples
- New Extensions view allows you to visually organize and order your installation
- InstallShield Installation Applet lets your users specify or get the appropriate Virtual Machine (VM), download, and install your Java application by clicking a Web link
- Self-extracting installations allow users to launch installation with point-and-click ease
- Wizard-style interface makes Java installation development a snap
- Specify required Java Virtual Machines; installation only runs on end-user systems with specified VMs
- Command-line access allows you to add installations to your batch file building process
- Launch scripts let your users launch your application with a one-word command
- Uninstallation feature, provision for silent mode installations, and much, much more

©1998 InstallShield Software Corporation. All rights reserved. InstallShield is a registered trademark and service mark of InstallShield Corporation. "It All Starts Here" is a service mark and a registered trademark of InstallShield Corporation. Windows is a registered trademark of Microsoft Corporation. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and/or other countries. Part No. 181-20005 0898

It All Starts Here®

900 National Parkway, Suite 125
Schaumburg, IL 60173-5108 USA
Sales: +1-847-240-0618
Fax: +1-847-619-0788
Toll-free: +1-800-440-8149
sales@installshield.com
www.installshield.com

InstallShield®
Software Corporation

www.installshield.com/java

It won't mean a thing If it ain't got that swing

JCLASS JavaBeans

You need ***powerful components*** for professional Java development. But if they don't work with Swing and the latest JDK 1.2 release... they won't mean a thing.

JClass, the world's most popular suite of 100% Pure JavaBeans, now includes ***JClass SwingSuite***, the essential set of extensions and enhancements for Swing in JDK 1.2. And with the power of JClass Suites, you'll be building for the future ***today.***

JCLASS WORKS WITH:

☑ JDK 1.0.2 (AWT)	☑ IBM VisualAge for Java	☑ Progress Apptivity
☑ JDK 1.1 (AWT or Swing)	☑ IST Visaj	☑ SilverStream
☑ JDK 1.2 (Swing)	☑ Microsoft Visual J++	☑ Sun Java WorkShop
☑ Bluestone Sapphire/Web	☑ NetBeans	☑ SuperCede
☑ Borland JBuilder	☑ Powersoft PowerJ	☑ Symantec Visual Café

New **JClass SwingSuite** - easily add the power of MDI, wizards and a robust collection of thread-friendly, lightweight components including JCTreeExplorer, JCTreeTable, JCSpinBoxes, a configurable DateChooser control and more!

- **JClass LiveTable** - the high performance Java grid/table for static, dynamic or db-bound data
- **JClass Chart** - the ultimate in Java charting and graphing, designed for superior performance
- **JClass HiGrid** - the innovative RAD outline-grid for hierarchical dynamic data
- **JClass DataSource** - a robust, hierarchical, multiple-platform data source
- **JClass Field** - complete input and validation for popular data types including calendar

Need rock-solid, robust Java GUIs? Professional Java developers around the world have made the award-winning JClass family of components their choice. Where else will you find seamless IDE integration and support for all major JDK versions? *Try JClass today for great performance with an improved swing!*

EVALUATE/ORDER ONLINE www.klgroup.com

KL GROUP
Software Development Productivity

KL Group Inc. Toll-Free: 1-888-361-3264 Tel: 416-594-1026 Fax: 416-594-1919
KL Group Europe B.V. Tel: +31 (0)20 679 95 03 Fax: +31 (0)20 470 03 26
www.klgroup.com

* Databinding feature requires purchase of JClass DataSource, or use of Borland JBuilder Professional or C/S edition, or Symantec Café dbDE edition. JClass, JClass Chart, JClass DataSource, JClass Field, JClass HiGrid, JClass LiveTable and JClass SwingSuite are trademarks of KL Group Inc. All other product names are trademarks or registered trademarks of their respective owners. JR004

FOR FASTER SERVICE & MORE INFORMATION:

FAX
615.370.4845

E-MAIL
subscriptions@sigs.com

ON-LINE
www.javareport.com

CUSTOMER SERVICE
1.800.361.1279

SIGS PUBLICATIONS

JR-FEB-D

YES!

Enter my subscription at the term below. This is a risk-free offer. If I'm not completely satisfied I may cancel and get a prompt refund for the unmailed issues.

❑ 1 year—$39 (12 issues)

❑ **2 years—$62 (24 issues) a 20% savings**

○ **Check enclosed** (payable to ***Java™ Report)***

Charge my ○Visa ○Mastercard ○American Express ○ **Bill me**

Card # Exp. date

Signature

Name

Title

Company

Address

City

State/Province Zip/Postal Code Country

Phone Fax

E-mail ○ Check here to receive only SIGS' promotions via e-mail

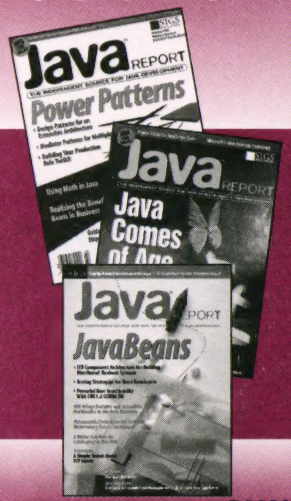
We reserve the right to rent our e-mail list

Make checks payable to ***Java™ Report***. Canadian and Mexican orders must add $25 per year. Checks must be in US dollars and drawn on a US bank. Canadian GST included (#R895884229)

Java and Java-based trademarks are registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Java™ Report is independent of Sun Microsystems, Inc.

Java™ REPORT
THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT

SUBSCRIPTION SAVINGS COUPON



B90203

PLACE
POSTAGE
HERE

Java™ REPORT

THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT

PO Box 5050

Brentwood, TN 37024-5050

FOR FASTER SERVICE & MORE INFORMATION:

FAX

615.370.4845

E-MAIL

subscriptions@sigs.com

ON-LINE

www.javareport.com

CUSTOMER SERVICE

1.800.361.1279

For Faster Service & More Information:

FAX
615.370.4845

E-MAIL
subscriptions@sigs.com

ON-LINE
www.sigs.com/publications/cppr/

CUSTOMER SERVICE
1.800.361.1279

SIGS PUBLICATIONS

C++ REPORT SUBSCRIPTION SAVINGS COUPON

YES!

Enter my subscription at the term below. This is a risk-free offer. If I'm not completely satisfied I may cancel and get a prompt refund for the unmailed issues.

❑ 1 year—$79 (10 issues)

❑ **2 years—$126** (20 issues) a 20% discount

❍ **Check enclosed** (payable to *C++ Report*)

Charge my ❍ Visa ❍ Mastercard ❍ American Express ❍ **Bill me**

Card # Exp. date

Signature

Name

Title

Company

Address

City

State/Province Zip/Postal Code Country

Phone Fax

E-mail ❍ Check here to receive only SIGS' promotions via e-mail

We reserve the right to rent our e-mail list

Make checks payable to *C++ Report*. Canadian and Mexican orders must add $25 per year.
Checks must be in US dollars and drawn on a US bank. Canadian GST included (#R895884229)

B90203

Place
Postage
Here

C++REPORT

PO Box 5050
Brentwood, TN
37024-5050

For Faster Service & More Information:

FAX
615.370.4845

E-MAIL
subscriptions@sigs.com

ON-LINE
www.sigs.com/publications/cppr/

CUSTOMER SERVICE
1.800.361.1279